# An aircraft based emulation platform and control model for LEO satellite antenna beam steering

by
Iwan Carel Kruger

*Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University*

Supervisor: Dr Riaan Wolhuter
Department of Electrical & Electronic Engineering

December 2010

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2010

# Abstract


## An Aircraft Based Emulation Platform and Control Module for LEO Satellite Antenna Beam Steering

I.C. Kruger

*Department of Electrical and Electronic Engineering,*
*University of Stellenbosch,*
*Private Bag X1, Matieland 7602, South Africa.*

Thesis: MScEng (Elec)

December 2010

A joint project between the KU Leuven and Stellenbosch Universities was at the time of this thesis underway to develop a space borne electronically beam steerable antenna and the associated ground-space segments. This thesis covers the development of an aircraft based satellite emulator to facilitate convenient aircraft based testing of an antenna array, intended for low earth orbit satellite deployment and subsystems to control the antenna array. A flight strategy is developed to emulate such a satellite pass as best possible, with the strategy implemented in software on in-flight PC hardware. A full interface between the aircraft avionics and satellite bus system has been developed to enable generation of the required antenna steering commands and to create a satellite bus image to the payload. Successful test results are presented, as obtained from the actual aircraft flight simulator. The thesis describes the successful development and testing of a low altitude flight test strategy for certain satellite borne systems, as a cost-effective and realistic interim step to actual and very expensive space flight testing.

# Uittreksel


## 'n Vliegtuig Gebaseerde Emulasie Platform en Beheer Module vir LEO Satelliet Antenna Straal Beheer

*("An Aircraft Based Emulation Platform and Control Module for LEO Satellite Antenna Beam Steering")*

I.C. Kruger

*Departement Elektriese en Elektroniese Ingenieurswese,*
*Universiteit van Stellenbosch,*
*Privaatsak X1, Matieland 7602, Suid Afrika.*

Tesis: MScIng (Elek)

Desember 2010

'n Gesamentlike projek deur KU Leuven en Stellenbosch Universiteit was tydens die verloop van hierdie tesis besig met die ontwikkeling om 'n ruimte gebaseerde elektroniese straal beheerde antenna en geassosieerde substelsels daar te stel. Hierdie tesis handel oor die ontwikkeling van 'n vliegtuig gebaseerde satelliet emulator om die toetsing van 'n elektroniese stuurbare antenna, wat bedoel is vir 'n lae aardse wentelbaan, te fasiliteer en die ontwikkeling van substelsels wat die stuurbare antenna beheer. 'n Vlug strategie is ontwikkel om so 'n satelliet wentelbaan so na as moontlik te emuleer. Die strategie word dan geïmplementeer in die sagteware van die aanboord vlug rekenaar. 'n Intervlak tussen die vliegtuig instrumente en satellietbus is ontwikkel om die generering van die nodinge instruksies te fasiliteer en om 'n virtuele satellietbus vir die res van die satelliet stelsel te skep. Suksesvolle toets resultate word getoon wat met behulp van 'n vliegtuig simulator verkry is. Die tesis beskryf die suksesvolle ontwikkeling en toetsing van 'n lae vlugtoets strategie vir satelliet stelsels, as 'n koste effektiewe en realistiese tussenstap, tot baie duur ruimte vlugtoetsing.

# Acknowledgements

I would like to express my sincere gratitude to the following people:

- My supervisor, Dr. R. Wolhuter, for his expert guidance.
- Members of the Leuven project for their input during the various stages of the project, especially Ewald van der Westhuizen for providing technical advice and support.
- My parents and sisters for their love and support.

**SOLI DEO GLORIA**

# Dedications

*In memory of my grandmother Maria van der Merwe*

# Contents

# List of Figures

# List of Tables

# Nomenclature

### Constants

| | | |
|---|---|---|
| $G = 6.672 \times 10^{-11}$ | Universal gravitation constant | $[\mathrm{m^3 kg^{-1} s^{-2}}]$ |
| $M_E = 5.974 \times 10^{24}$ | Earth mass | $[\mathrm{kg}]$ |
| $\pi =$ | 3.141 592 654 | |
| $c = 3 \times 10^8$ | Speed of light in free space | $[\mathrm{ms^{-1}}]$ |
| $k_B = 1.38 \times 10^{-23}$ | Boltzmann's constant | |
| $T_0 = 290$ | Reference temperature | $[\mathrm{K}]$ |

### Variables

| | | |
|---|---|---|
| $\varphi$ | Latitude | $[\mathrm{rad}]$ |
| $\lambda$ | Longitude | $[\mathrm{rad}]$ |
| $h$ | Altitude | $[\mathrm{km}]$ |
| $\Phi$ | Roll angle | $[\mathrm{rad}]$ |
| $\Theta$ | Pitch angle | $[\mathrm{rad}]$ |
| $\Psi$ | Yaw angle | $[\mathrm{rad}]$ |
| $r$ | Radius of circular orbit | $[\mathrm{m}]$ |
| $v$ | Orbit velocity | $[\mathrm{ms^{-1}}]$ |
| $S$ | Distance of one circular orbit | $[\mathrm{m}]$ |
| $T$ | Orbital period | $[\mathrm{s}]$ |
| $\omega$ | Angular velocity | $[\mathrm{s^{-1}}]$ |
| $D$ | Distance to satellite | $[\mathrm{km}]$ |
| $E$ | Elevation angle | $[\mathrm{rad}]$ |
| $L_{FS}$ | Free space loss | $[\mathrm{dB}]$ |
| $f$ | Frequency | $[\mathrm{Hz}]$ |

| | | |
|---|---|---|
| $P$ | Signal strength . . . . . | [dB] |
| $T$ | Noise temperature . . . . . | [K] |
| $B$ | Bandwitdh . . . . . | [Hz] |

**Subscripts**

| | |
|---|---|
| *sat* | Satellite |
| *GS* | Ground station |
| *tot* | Total |
| *ant* | Antenna |
| *rec* | Receiver |

**Acronyms**

| | |
|---|---|
| AAEE | Aircraft Avionics Equipment Emulator |
| ADC | Analogue-to-Digital Converter |
| ASE | Aircraft Satellite Emulator |
| BER | Bit Error Rate |
| CAN | Controller Area Network |
| DBS | Digital Beam Steering |
| DLL | Dynamic Link Library |
| DTR | Data-Terminal-Ready |
| ECEF | Earth Centred Earth Fixed |
| E-field | Electric Field |
| FID | Frame Identifier |
| FPGA | Field Programmable Gate Array |
| FSL | Free Space Loss |
| GUI | Graphical User Interface |
| ID | Identifier |
| IPC | Inter Process Communication |
| LEO | Low Earth Orbit |
| LLA | Latitude, Longitude and Altitude |
| NED | North-East-Down |

| | |
|---|---|
| OBC | On-Board Computer |
| PC | Personal Computer |
| PLL | Phase Locked Loop |
| POSIX | Portable Operating System Interface for Unix |
| QNX | Quick Unix |
| RF | Radio Frequency |
| RTS | Request-To-Send |
| RX | Receive |
| SAA | Steerable Antenna Array |
| SNR | Signal to Noise Ratio |
| UART | Universal Asynchronous Receiver/Transmitter |
| UML | Unified Modelling Language |
| WGS-84 | World Geodetic System 1984 |

# Chapter 1

# Introduction

## 1.1 Project Background

The ESAT-TELEMIC division of the Department of Electrical Engineering, of Katholieke Universiteit Leuven, Flanders, was at the time of this thesis doing research in advanced techniques for the design of electronically beam steerable antenna arrays (SAA), intended for deployment in space on satellites [1]. One of the objectives of such research is to enable the deployment of relatively low cost ground stations for environmental and agricultural data acquisition. By introducing beam steered satellite antenna tracking of ground stations during overflight, the link budget could be improved and ground station complexity reduced, particularly with regard to antenna design and the RF chain.

As a partner to this undertaking, The Department of Electrical and Electronic Engineering of Stellenbosch University, South Africa, was commissioned to develop the rest of the satellite payload, the ground station and the accompanying ground-space communications link. The payload is intended for a low earth orbit (LEO) satellite. Development and construction of any form of space borne system is normally expensive and associated with many risks. It was, therefore, decided to introduce an interim step prior to actual space flight, by using a light aircraft as a pseudosatellite test platform, providing the obvious advantages of convenient and relatively cheap system testing and debugging and clearly enhancing the chances of eventual in-flight success. The aircraft itself has been fitted out for experimental use and will house the SAA in an external pod, the entire satellite payload containing the On Board Computer (OBC), communications link component chain, steerable antenna con-

trol/status interface, power supplies, as well as an Aircraft Satellite Emulator (ASE). In actual deployment, all interaction with the rest of the satellite is via a system bus for purposes of telecommand, telemetry and attitude/positioning information. The ASE is obviously required to act as translator and emulator between the payload and aircraft, the latter acting as pseudo-satellite. As far as the payload is concerned, it should behave as if connected to the actual satellite bus. The ASE and developed emulation strategy can be adapted to various LEO satellite payloads and thus provides a general low cost test platform prior to space deployment.

The purpose of this thesis is to report on the development of the emulation platform and SAA control subsystems, and to present some very encouraging test results. The thesis will describe the required flight path mechanics, the feasibility of an emulation strategy, a description of the implementation and test results obtained from an actual aircraft simulator.

## 1.2 Objectives

The primary objectives set at the inception of this project are as follows:

- Develop an aircraft borne LEO satellite emulation package for the purpose of testing a satellite payload containing a SAA.
- Design the emulation system to provide the necessary data to the payload, thereby simulating an actual satellite.
- Development of the SAA-payload interface.
- Development of a control algorithm for the payload OBC to control the SAA. The control algorithm directs the SAA to a specific location.
- Test and evaluate the emulation system and control algorithm.

## 1.3 Development Approach

A systematic development approach has been followed. A background study covering subjects fundamental to the system was first be completed. This included an investigation into the behaviour of LEO satellites and the identification of orbital parameters crucial to successful emulation of a LEO satellite.

Once the background study has been completed, the next step addressed the emulation strategy.

The emulation strategy investigated and presented methods enabling an airborne platform to emulate orbital characteristics of the identified LEO satellite as closely as possible.

The next stage of the development entailed the system design of the project. The system architecture and system-wide design decisions were covered in this phase.

The system design was followed by the system implementation, covering and identifying critical components, subsystems and their implementation.

The final phase of the development evaluated and thoroughly tested the system and documented the ensuing results thereof .

## 1.4 Summary of Contributions

The contributions of this work, can be summarised as follows:

- The creation of a basic LEO satellite orbital model.
- Development of an emulation strategy for aircraft borne flight testing of selected satellite payloads.
- Development of the required emulation system.
- Development of the SAA control algorithm.
- The implementation of the developed system.
- Ground based evaluation of the system using flight simulators to prove and demonstrate functionality and feasibility of the system.

## 1.5 Structure of Thesis

The structure of this thesis is as follows:

Chapter 2 provides the necessary background to the development of the system. The chapter starts by investigating the reference frames used and the transformation between them. The chapter then proceeds to the attitude

description of an airborne object. The orbital calculations for, and characteristics of a LEO satellite are then analysed. The chapter then presents a brief overview of the hardware used and concludes with an overview of factors affecting the link budget.

The emulation strategy is developed in Chapter 3. This strategy comprises two parts, namely the flight strategy and the transmission link strategy. The thesis then proceeds to the system design phase of the project.

Chapter 4 discusses the system design, which starts with a functional analysis followed by a UML use case and sequence diagram analysis of the system. Thereafter, the design of the main system components and the functional units contained in each component are described in more detail.

Chapter 5 presents the system implementation, which starts with the emulator module and then proceeds to the control module implementation.

Various tests were performed to evaluate the system. These tests are presented and discussed in Chapter 6. The chapter ends with a simulated flight test, verifying the performance of the system.

The thesis ends with Chapter 7, containing conclusions and summaries as well as proposals for future work.

# Chapter 2

# Background

This chapter discusses the various concepts required to develop the system. The reference frames used and the conversion between them are first discussed. Thereafter a review is given of the Euler angles, as required to describe the orientation of an airborne object. A brief review of orbital calculations is then presented. These calculations form an integral part of the system and the orbital model created in this chapter. The model investigates the orbital characteristics of LEO satellite. The chapter then discusses the CAN bus and SAA. These two hardware components form a critical part of the project. The chapter finally concludes with an overview of the factors affecting the link budget.

## 2.1 Body Reference Frame

The body reference frame described here will be used to define the orientation of the satellite and the aircraft. See Figure 2.1 and 2.2 for an illustration.

The right handed reference frame is situated at the aircraft's centre of gravity. The x-axis is directed from the frame origin towards the nose of the aircraft, the z-axis downward and the y-axis along the starboard wing.

This reference frame is also applicable to a satellite, in which case instead of pointing the x-axis towards the aircraft nose, the x-axis is pointed towards the "satellite nose", meaning the direction of forward motion of the satellite.

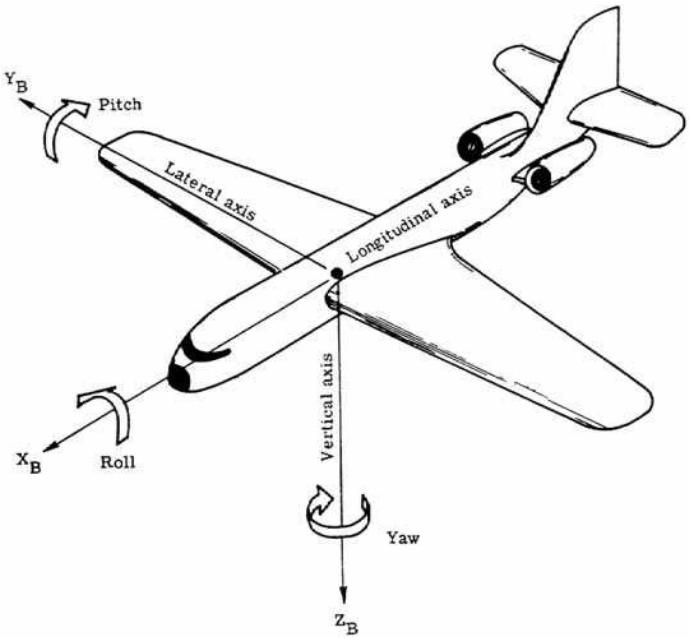


Figure 2.1: Aircraft body axes definition [2]

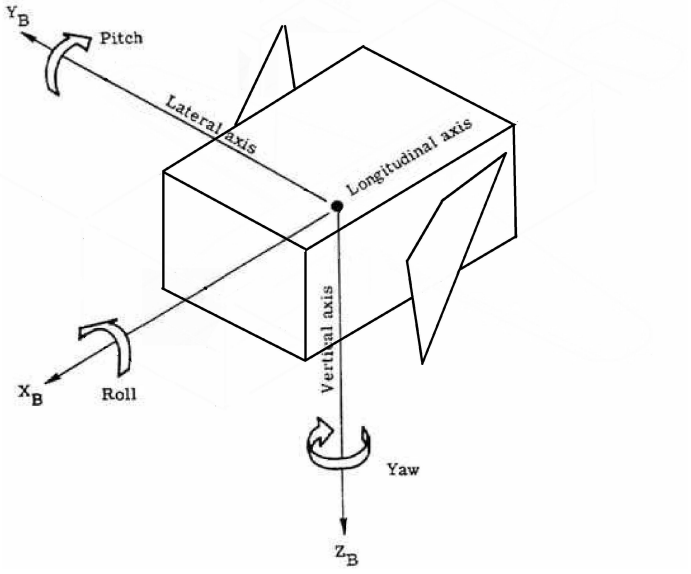


Figure 2.2: Satellite body axes definition

## 2.2 ECEF Reference Frame

The right handed Earth Centred Earth Fixed (ECEF) frame is attached to the Earth, with its centre coinciding with the centre of the Earth. The frame

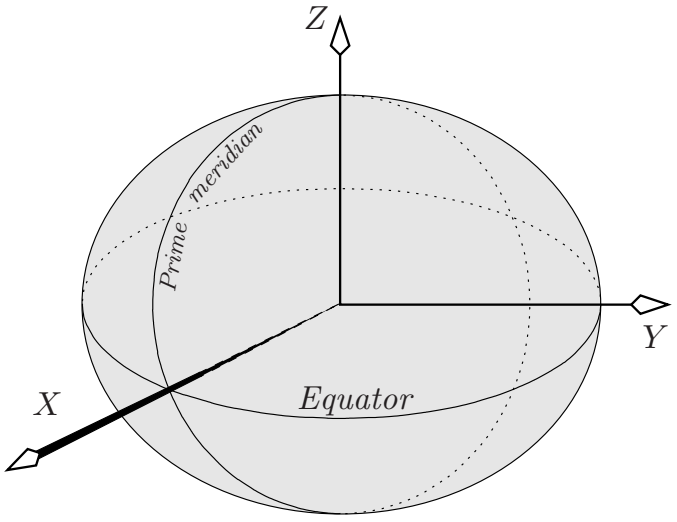


Figure 2.3: ECEF reference frame [3]

is non-inertial and thus rotates with the Earth. The x-axis points towards the intersection of the Prime meridian and the Equator. The y-axis is perpendicular to the x-axis on the equatorial plane and the z-axis points towards the North pole. Cartesian and LLA (latitude, longitude and altitude) coordinate systems are both commonly used to describe a location on this frame.

A location is often described by the LLA coordinate system; however, calculations are easier to perform using Cartesian coordinates. It is thus necessary to convert LLA coordinates to Cartesian coordinates.

The following conversions are derived by using the World Geodetic System 1984 standard (WGS-84) describing the shape of the Earth as an oblate spheroid. WGS-84 defines the Earths semi-major axis $a = 6378.137$ km and semi-minor axis $b = 6356.7523142$ km.[4]

$$X = \left( \frac{a}{\sqrt{\cos^2(\varphi) + \frac{b^2}{a^2} \cdot \sin^2(\varphi)}} + h \right) \cdot \cos(\varphi) \cos(\lambda) \quad (2.2.1)$$

$$Y = \left( \frac{a}{\sqrt{\cos^2(\varphi) + \frac{b^2}{a^2} \cdot \sin^2(\varphi)}} + h \right) \cdot \cos(\varphi) \sin(\lambda) \quad (2.2.2)$$

$$Z = \left( \frac{b}{\sqrt{\frac{b^2}{a^2} \cdot \cos^2(\varphi) + \sin^2(\varphi)}} + h \right) \cdot \sin(\varphi) \quad (2.2.3)$$

with geodetic latitude $\varphi$, longitude $\lambda$ and altitude $h$.

## 2.3 NED Reference Frame

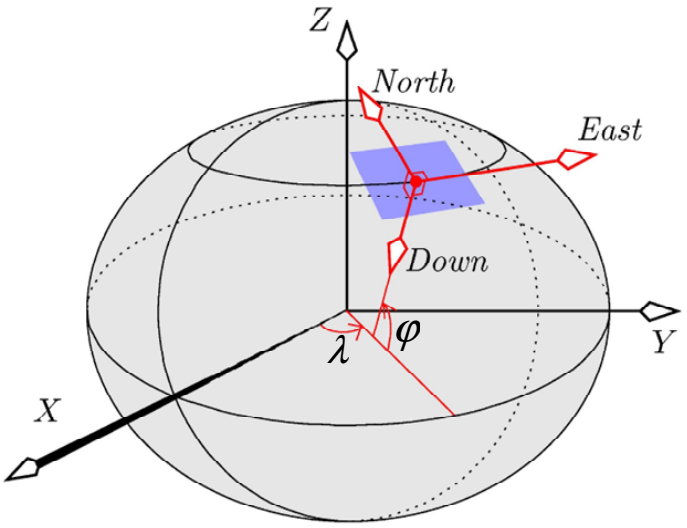


Figure 2.4: NED reference frame [5]

North-East-Down (NED) frame is a local geographic plane and can be constructed at any convenient point. The axes of this frame always point North, East and down towards the surface of the Earth. For the purpose of this project this frame is constructed at the position of the airborne object, with the frames centre coinciding with the centre of the body axes of the airborne object. The NED frame is used in this project as an orbit-defined reference frame to describe the attitude of the airborne object.

## 2.4 Convert from ECEF Frame to NED Frame

A vector coordinate defined as $\vec{V}_{ECEF} = \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix}^T$ in the ECEF frame, is converted to a vector coordinate $\vec{V}_{NED} = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T$ in the NED frame by multiplying the $\vec{V}_{ECEF}$ vector with the transformation matrix $\mathbf{K}$. The transformation matrix is defined as: [6]

$$\mathbf{K} = \begin{bmatrix} -\sin(\varphi)\cos(\lambda) & -\sin(\varphi)\sin(\lambda) & \cos(\varphi) \\ -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\varphi)\cos(\lambda) & -\cos(\varphi)\sin(\lambda) & -\sin(\varphi) \end{bmatrix} \tag{2.4.1}$$

where $\varphi$ and $\lambda$ are respectively, the latitude and longitude.

The transformation is summarised as:

$$\vec{V}_{NED} = \mathbf{K} \cdot \vec{V}_{ECEF} \tag{2.4.2}$$

## 2.5 Euler Angles

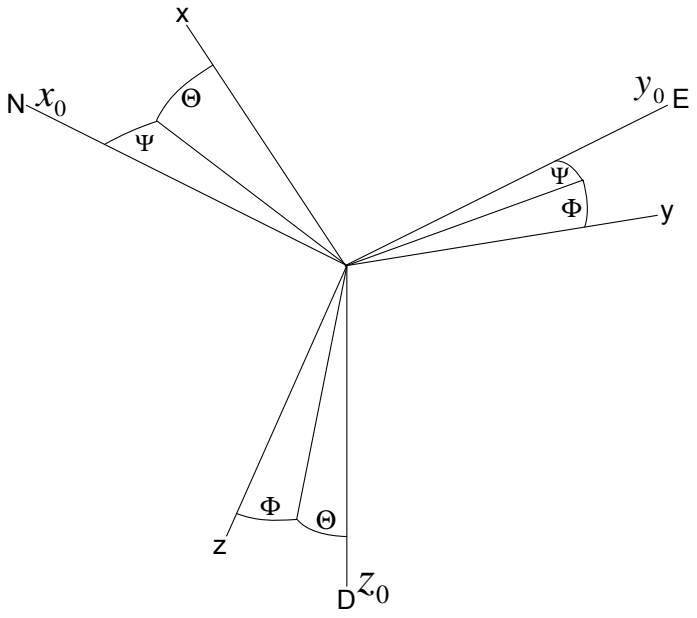


Figure 2.5: Euler angles measured from the NED reference frame

Euler angles, known as roll $\Phi$, pitch $\Theta$ and yaw $\Psi$ angles, describe the orientation of an object. These angles are measured from an orbit-defined

reference frame towards the body axis $F(xyz)$. Therefore, if the roll, pitch and yaw angles are all zero, the body frame is perfectly aligned with the orbit-defined reference frame. See Figure 2.5 for an illustration of these Euler angles.

The Euler 3-2-1 method was adopted to perform coordinate transformations from the NED coordinate frame to the body coordinate frame, which implies that the rotations are performed in the sequence of yaw, pitch and roll. Thus a vector described in the NED reference frame can be expressed in the body reference frame after these rotations have been performed. The following describes the Euler 3-2-1 alignment of the NED reference frame to the body reference frame. The aim of this alignment is to create a transformation matrix, that will transform coordinates from the NED frame to the body frame. See Figure 2.6 for a graphical representation.

- Define frame $F_0(x_0y_0z_0)$ that is aligned with the NED reference frame.
- Rotate $F_0$ with yaw angle $\Psi$, about the $z_0$ axis with $F_1(x_1y_1z_1)$ as a result.

$$F_1 = \mathbf{R}_z(\Psi)F_0$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{2.5.1}$$

- Rotate $F_1$ with pitch angle $\Theta$, about the $y_1$ axis with $F_2(x_2y_2z_2)$ as a result.

$$F_2 = \mathbf{R}_y(\Theta)F_1$$

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{2.5.2}$$

- Rotate $F_2$ with roll angle $\Phi$, about the $x_2$ axis. The result is the alignment of the NED reference frame to the body reference frame.

$$F = \mathbf{R}_x(\Phi)F_2$$

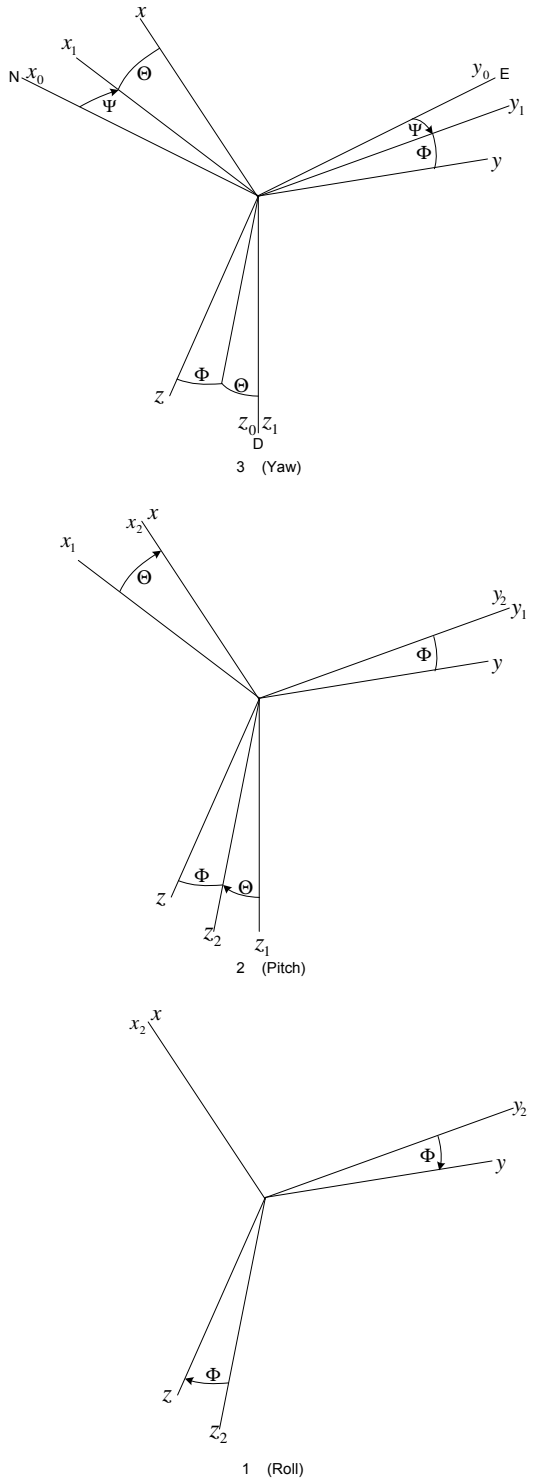


Figure 2.6: Euler 3-2-1 transformation method

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{2.5.3}$$

A Direction Cosine Matrix (DCM) combines all three rotations.

$$\mathbf{A} = \mathbf{R}_x(\Phi)\mathbf{R}_y(\Theta)\mathbf{R}_z(\Psi) \tag{2.5.4}$$

$$= \begin{bmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{bmatrix} \tag{2.5.5}$$

The defined matrix $\mathbf{A}$ is thus a transformation matrix and is used to transform a vector ($V_N$) defined in the NED reference frame to a vector ($V_B$) in the body reference frame.[7]

$$\vec{V_B} = \mathbf{A}\vec{V_N} \tag{2.5.6}$$

## 2.6 Orbital Calculations

This section presents a brief summation of the satellite orbital mechanics, as these are fundamental to the system design. The calculations presented in this section are based on a spherical earth model, which is adequate for this particular type of application. The oblateness of the earth and the varying topography on the surface, are treated as coordinates above or below the spherical surface of the earth.[8]

### 2.6.1 Orbital Velocity

Newton's second law of motion states that the net forces acting on an object are directly proportional to its acceleration and mass. From Newton's second law and the fact that an object in a circular orbit experience centripetal acceleration, we obtain

$$F = m\frac{v^2}{r} \tag{2.6.1}$$

where r (m) is the radius of the circular orbit. The only external force acting on an object in orbit is the gravitational force. Therefore reduce (2.6.1) to

$$G\frac{M_E m}{r^2} = m\frac{v^2}{r} \tag{2.6.2}$$

where $G = 6.672 \times 10^{-11}\,\mathrm{m^3kg^{-1}s^{-2}}$ is the universal gravitation constant and the mass of the earth $M_E = 5.974 \times 10^{24}\,\mathrm{kg}$.[9] From (2.6.2) the velocity is defined as

$$v = \sqrt{\frac{GM_E}{r}} \qquad (ms^{-1}) \tag{2.6.3}$$

Thus the velocity of an object in circular orbit is determined by its altitude. Figure 2.7 shows the satellite velocity at various altitudes above the earth.

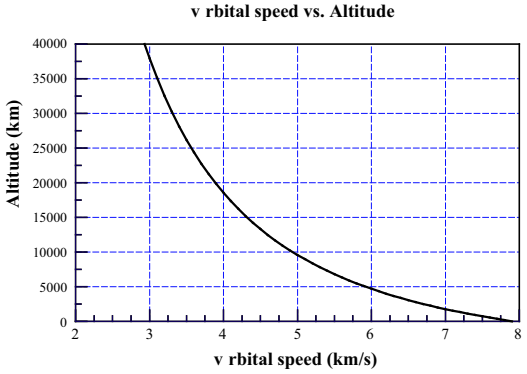


Figure 2.7: Satellite orbital speed [10]

### 2.6.2 Orbital Period

The orbital period of a satellite in a circular orbit can be calculated as

$$T = \frac{S}{v} \qquad (s) \tag{2.6.4}$$

$$= \frac{2\pi r}{v} \tag{2.6.5}$$

where $S$ is the distance the satellite travels to complete one circular orbit. The distance $S$ is calculated by using the relationship between the circumference $S$ and the radius of a circle, where r (m) is the radius of the circular orbit. Substituting the velocity ($v$) with Equation 2.6.3, the orbital period can be calculated as

$$T = 2\pi\sqrt{\frac{r^3}{GM_E}} \qquad (s) \tag{2.6.6}$$

### 2.6.3 Angular Velocity

The angular velocity of a satellite in orbit can be calculated by

$$\omega = \frac{2\pi}{T} \qquad (s^{-1}) \tag{2.6.7}$$

By use of Equation 2.6.6, the angular velocity is found to be

$$\omega = \sqrt{\frac{GM_E}{r^3}} \qquad (s^{-1}) \tag{2.6.8}$$

### 2.6.4 Coordinates

The locations of the satellite and ground stations are specified in latitude, longitude and radius coordinates, and can be expressed in celestial coordinates originating at the earth's centre, as shown in Figure 2.8.

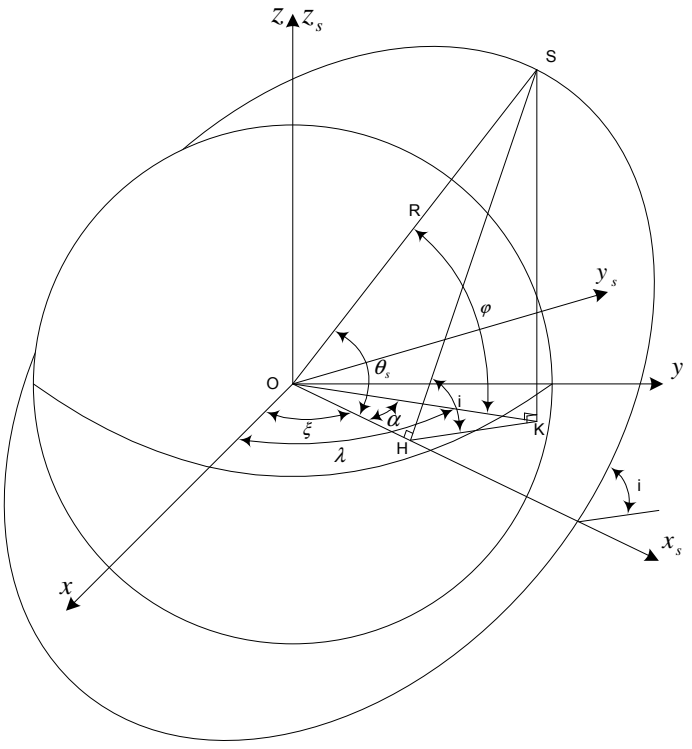


Figure 2.8: Celestial coordinate structure

### Satellite coordinates

The satellite position is first described by the $(x_s, y_s, z_s)$ coordinate system (Figure 2.8). The $x$-axis is directed to the intersection of the satellite orbital path and the equatorial plane. The satellite coordinates in this coordinate frame are:

$$x_s = R\cos(\theta_s) \tag{2.6.9}$$

$$y_s = R\sin(\theta_s)\cos(i) \tag{2.6.10}$$

$$z_s = R\sin(\theta_s)\sin(i) \tag{2.6.11}$$

where $i$ is the inclination angle and $\theta_s = \theta_0 + (\omega_{sat})(t)$ the orbit angle. The initial orbit angle can be calculated by:

$$\theta_0 = \arcsin\left(\frac{\sin(\varphi_0)}{sin(i)}\right) \tag{2.6.12}$$

where $\varphi_0$ is the initial geocentric latitude coordinate of the satellite and $\omega_{sat}$ the angular velocity.

A transformation is required to describe the $(x_s, y_s, z_s)$ coordinates in a celestial coordinate system. The transformation is presented in the following matrix notation:

$$\begin{bmatrix} x_{sat} \\ y_{sat} \\ z_{sat} \end{bmatrix} = \begin{bmatrix} \cos(\xi) & \sin(\xi) & 0 \\ -\sin(\xi) & \cos(\xi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} \tag{2.6.13}$$

The $\xi$ angle is obtained as $\xi = \alpha - \lambda_{Sat,0}$ where $\lambda_{Sat,0}$ is the initial longitude coordinate of the satellite. $\alpha$ can be calculated as:

$$\alpha = \arccos\left(\frac{\cos(\theta_0)}{cos(\varphi_0)}\right) \tag{2.6.14}$$

Thus, the satellite coordinates are given by:

$$x_{Sat} = R\cos(\theta_s)\cos(\xi) + R\sin(\theta_s)\cos(i)\sin(\xi) \tag{2.6.15}$$

$$y_{Sat} = -R\cos(\theta_s)\sin(\xi) + R\sin(\theta_s)\cos(i)\cos(\xi) \tag{2.6.16}$$

$$z_{Sat} = R\sin(\theta_s)\sin(i) \tag{2.6.17}$$

**Ground station coordinates**

From Figure 2.8 the ground station coordinates, transformed to the celestial coordinate system, are defined by:

$$x_{GS} = R_{GS}\cos(\varphi_{GS})\cos(\lambda_{GS}) \quad (2.6.18)$$

$$y_{GS} = R_{GS}\cos(\varphi_{GS})\sin(\lambda_{GS}) \quad (2.6.19)$$

$$z_{GS} = R_{GS}\sin(\varphi_{GS}) \quad (2.6.20)$$

with $R_{GS}$ the distance of the ground station from the centre of the earth and $\varphi_{GS}$ the ground station geocentric latitude coordinate. $\lambda_{GS}(t) = \lambda_{GS,0} + (\omega_{GS})(t)$, where $\lambda_{GS,0}$ is the initial longitude coordinate of the ground station and $\omega_{GS}$ the angular velocity of the Earth. The earth takes one sidereal day to complete one rotation.[11] Thus the angular velocity of the earth is calculated as follows:

$$\omega_{GS} = \frac{2\pi}{1 \text{ sidereal day}} = \frac{2\pi}{86164} \quad (2.6.21)$$

$$= 7.292e{-}5 \qquad (s^{-1}) \quad (2.6.22)$$

### 2.6.5 Distance to satellite

The distance between the ground station and the satellite clearly varies with the satellite orbit. If the coordinates of the satellite and ground station are known, the distance $D$ between the ground station and the satellite can be obtained quite simply by Pythagorean geometry.

$$D = \sqrt{(x_{sat} - x_{GS})^2 + (y_{sat} - y_{GS})^2 + (z_{sat} - z_{GS})^2} \quad (2.6.23)$$

The distance can also be calculated as a function of the elevation angle $E$. The elevation angle varies from 0° at the horizon to 90° when the satellite is directly above the ground station. Referring to the triangle created by the satellite, ground station and earth centre in Figure 2.9, the satellite to ground station distance can be calculated by the cosine rule. The earth radius $R$ and altitude $h$ are known.

$$(h+R)^2 = D^2 + R^2 - 2 \cdot R \cdot D \cdot \cos(90° + E) \quad (2.6.24)$$

$$0 = D^2 - 2 \cdot R \cdot \cos(90° + E) \cdot D - (h+R)^2 + R^2 \quad (2.6.25)$$

$$D = \frac{2 \cdot R \cdot \cos(90° + E) + \sqrt{(2 \cdot R \cdot \cos(90° + E))^2 - 4 \cdot (R^2 - (h+R)^2)}}{2} \quad (2.6.26)$$

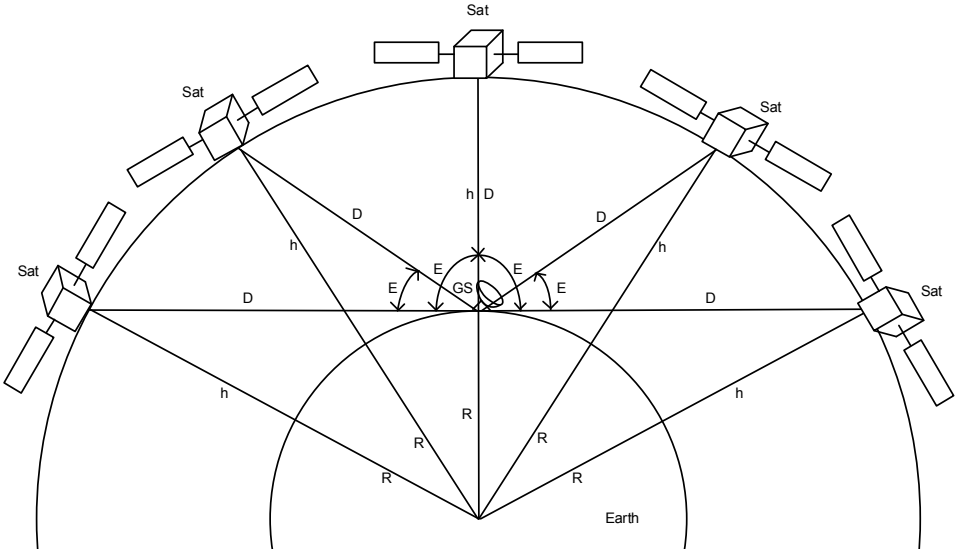


Figure 2.9: Satellite to ground station geometry

### 2.6.6 Geocentric angle

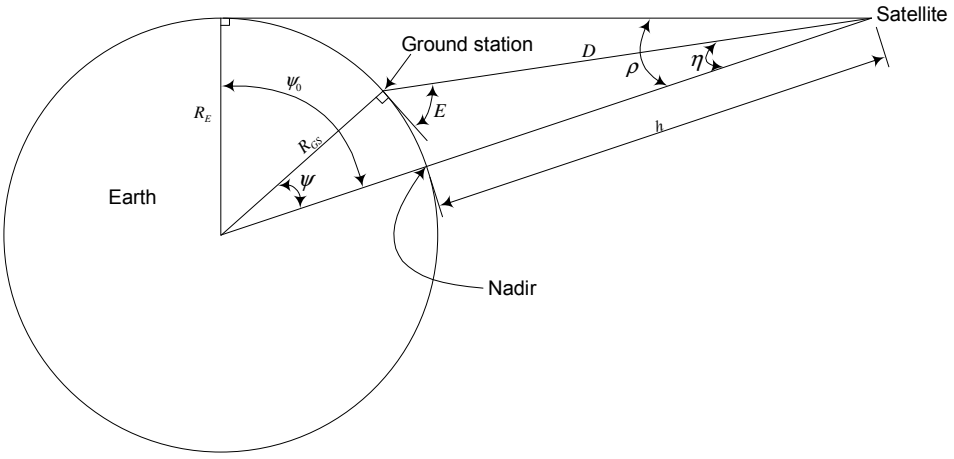


Figure 2.10: Angular relationships between satellite, ground station and earth centre [12]

The geocentric angle ($\psi$) is the angle measured at the earth's centre between the satellite nadir point and a ground station. Referring to Figure 2.10, the $\psi$ angle is determined by the cosine rule.

$$\psi = \arccos\left(\frac{R_{GS}^2 + R_{sat}^2 - D^2}{2 \cdot R_{GS} \cdot R_{sat}}\right) \tag{2.6.27}$$

### 2.6.7 Elevation angle

The Elevation angle ($E$) is the angle between the horizon and the satellite.

$$E = \left| \arcsin\left(\frac{R_{sat}\sin(\psi)}{D}\right) - \frac{\pi}{2} \right| \tag{2.6.28}$$

A satellite pass with a high maximum elevation angle will pass more or less directly above a ground station, whereas a satellite pass with a low maximum elevation angle will move across the horizon, from the perspective of the ground station. A satellite pass with a high maximum elevation angle constitutes good pass. The reason for this is that the time the satellite and ground station see each other, increases as the maximum elevation angle increases. Therefore, the communication time is longer. The direct line of sight distance between the satellite and ground station is also shorter at higher elevation angles. The shorter distance contributes to a lower free space loss, which enables a better transmission link.

### 2.6.8 Azimuth angle

The azimuth angle is the angle measured Eastward from North, to the nadir point at the ground station, as shown by angle NPT in Figure 2.11.

For the spherical triangle NPT of Figure 2.11

$$\frac{\sin(NPT)}{\sin(90^\circ - \varphi_{sat})} = \frac{\sin(NPT)}{\cos(\varphi_{sat})} = \frac{\sin(PNT)}{sin(\psi)} \tag{2.6.29}$$

For the spherical triangle NBA the angles BAN and AON are equal to 90°, therefore

$$\frac{\sin(BNA)}{\sin(L)} = \frac{\sin(BAN)}{\sin(AON)} = 1 \tag{2.6.30}$$

Because the angle BNA = PNT, Equation 2.6.30 can be substituted into Equation 2.6.29 with the result:

$$\sin(NPT) = \frac{\sin(L)\cos(\varphi_{sat})}{\sin(\psi)} \tag{2.6.31}$$

$$a = NPT = \arcsin(\frac{\sin(L)\cos(\varphi_{sat})}{\sin(\psi)}) \tag{2.6.32}$$

where $\psi$ is the geocentric angle, $\varphi_{sat}$ is the latitude coordinate of the satellite and

$$L = |\lambda_{GS} - \lambda_{sat}| \tag{2.6.33}$$

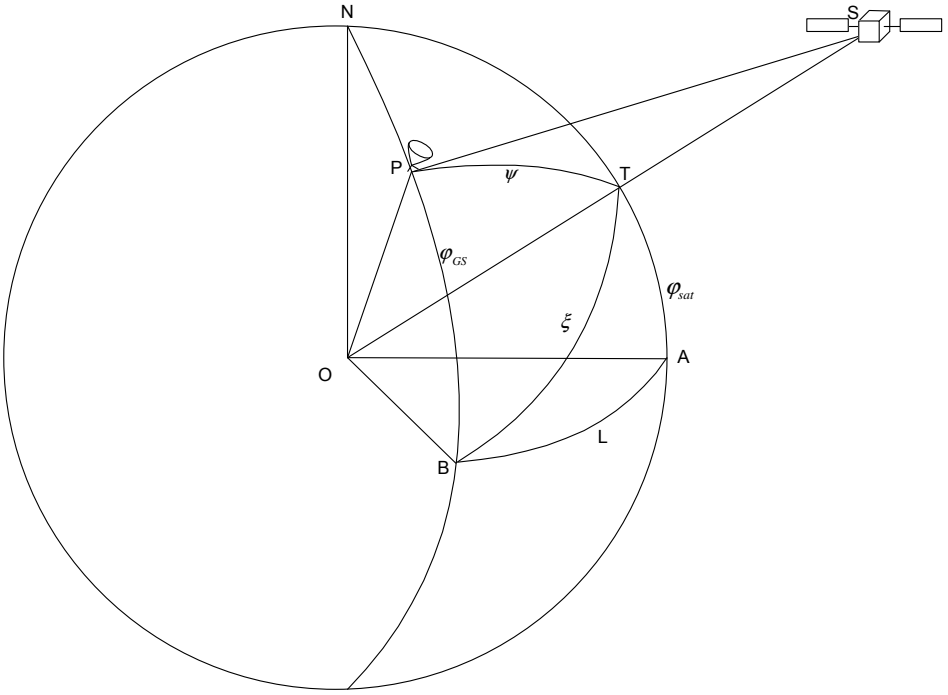


Figure 2.11: Earth satellite geometry [13]

the difference between the longitude coordinates of the ground station and the satellite. To obtain the true azimuth angle (A), we need to consider the position of the nadir point (T in Figure 2.11) relative to the ground station (point P in Figure 2.11). Therefore, it must first be determined in which quadrant the nadir (sub-satellite) point is with respect to the ground station and thereafter what the relationship between A and a. The four quadrants are situated South-East, North-East, South-West and North-West of the ground station. The various cases can be summarised as follows:

$$A = \begin{cases} 180° - a & \text{if } \lambda_{GS} - \lambda_{sat} > 0 \text{ and } \varphi_{GS} - \varphi_{sat} < 0, \text{ South-East} \\ a & \text{if } \lambda_{GS} - \lambda_{sat} > 0 \text{ and } \varphi_{GS} - \varphi_{sat} > 0, \text{ North-East} \\ 180° + a & \text{if } \lambda_{GS} - \lambda_{sat} < 0 \text{ and } \varphi_{GS} - \varphi_{sat} < 0, \text{ South-West} \\ 360° - a & \text{if } \lambda_{GS} - \lambda_{sat} < 0 \text{ and } \varphi_{GS} - \varphi_{sat} > 0, \text{ North-West} \end{cases} \tag{2.6.34}$$

## 2.7 Orbital Characteristics

A satellite-earth model was created in Matlab by using the orbital calculations presented in the previous section. The aim of the model is to investigate the orbital characteristics of a LEO polar orbit and the implications of such a orbit. In essence the model describes two objects rotating with different velocities in a circular motion. See Figure 2.12 for a graphical representation of these two orbits. By simulating the position of the satellite and ground station relative to each other in time, it is possible to perform various calculations. This section first discusses the Matlab model configuration. This includes the initial start up parameters assigned in the model. Then, finally, the calculations performed in the model and their interpretation.

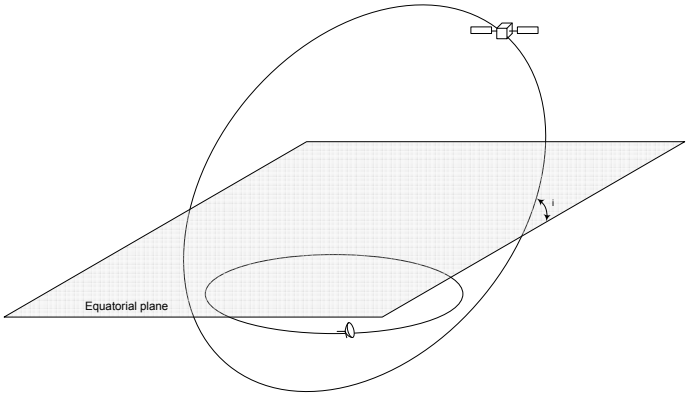


Figure 2.12: Satellite and ground station orbits

### 2.7.1 Model Configuration

The parameters required by the calculations presented in section 2.6 must first be initialised. These calculations are used by the model to simulate the relationship between a specific satellite in orbit and a ground station. The model configuration is as follows:

- A LEO altitude of 500 km is assigned. This is the altitude at which this specific satellite would operate. Note, as shown in section 2.6.2, that the orbital period is dependant on the altitude.

- A LEO path with a high inclination angle was chosen, meaning that the angle between the intersection of the orbital path and the equatorial plane is almost 90°. See Figure 2.12. Therefore the satellite will orbit over the poles. In polar orbit with the earth rotating beneath, the satellite covers the whole earth.

- The location of the ground station was chosen at coordinates in South Africa. The reason for this location is that at the time of the implementation of the Matlab simulation model it was unclear where the ground stations would be situated. The probability was, however, high that one would be situated in South Africa or at a similar latitude from the equator.

- The final configuration of the Matlab model chooses an arbitrary starting position for the satellite. This position is given by the initial satellite coordinates when the simulation starts. However, note that the satellite coordinates will change over time as the satellite orbits the earth.

### 2.7.2 Simulation

The Matlab model simulated satellite passes of a few weeks. Various calculations were made during this simulation as presented in this section.

Figure 2.13 shows the elevation angles calculated for a simulation of a few weeks, but displays the elevation angles in a time window of 20 days. The following observations were made regarding this figure:

- From the figure it is clear that the satellite and ground station will see each other four times in a period of 24 hours. The reason for this is as follows. A LEO satellite at an altitude of 500 km has an orbital period of approximately 94 minutes. The earth takes one sidereal day to complete one rotation. The satellite orbital period is therefore much shorter than one earth rotational period. From the time of the satellite's first pass it is possible for the satellite to complete another pass before the earth would have rotated enough for the satellite and ground station to be unable to see each other. This implies that if the ground station sees the satellite twice, moving from North to South across it, the ground station will see the satellite again approximately 12 hours later when the satellite moves

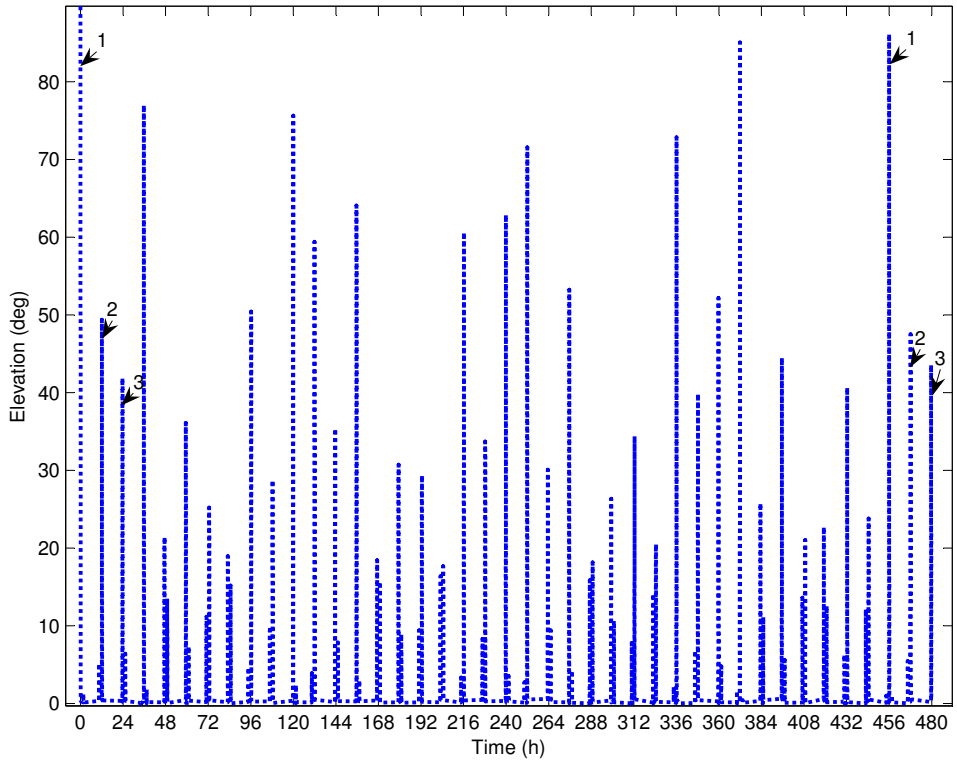


Figure 2.13: Elevation angle calculated for a period of 20 days

from South to North across it. Therefore the path of the satellite and ground station will cross twice at intervals of 12 hours.

- The observation is also made that if a very good pass occurs, the next pass approximately 94 minutes later will be very bad, and vice versa. This is attributable to the rotation of the earth.

- It is evident from the Figure 2.13, that most satellite passes will not be directly above the ground station. Most passes will have a lower maximum elevation angle. Thus the advantage of having a steerable antenna is evident. Steering an antenna beam will drastically improve the communication time as well as the transmission link.

- By investigating the simulated elevation and azimuth angles calculated for a few weeks, it was found that the satellite will almost be in exactly the same place with respect to the ground station after 19 days (456 hours), thus repeating the orbital relationship between the satellite and the ground station. Note, however, that the satellite will not be in pre-

cisely the same spot as it was in 19 days earlier. Stated differently, this cycle of 19 days created by the rotation of the earth and satellite does not create a perfect cycle, but is, however, close. By examining Figure 2.13, this cycle can be seen. The elevation angles will start to repeat after 19 days (456 hours). The elevation angles in the new cycle do not exactly match those of the previous cycle. There will always be a small difference. The reason for this is that the satellite is not in exactly in the same location with respect to the ground station. This cycle is, however, useful to get an indication of the typical elevation angles. The cycle can further be divided into two parts. These two situations occur when the satellite moves first from North to South, and then from South to North across the ground station.

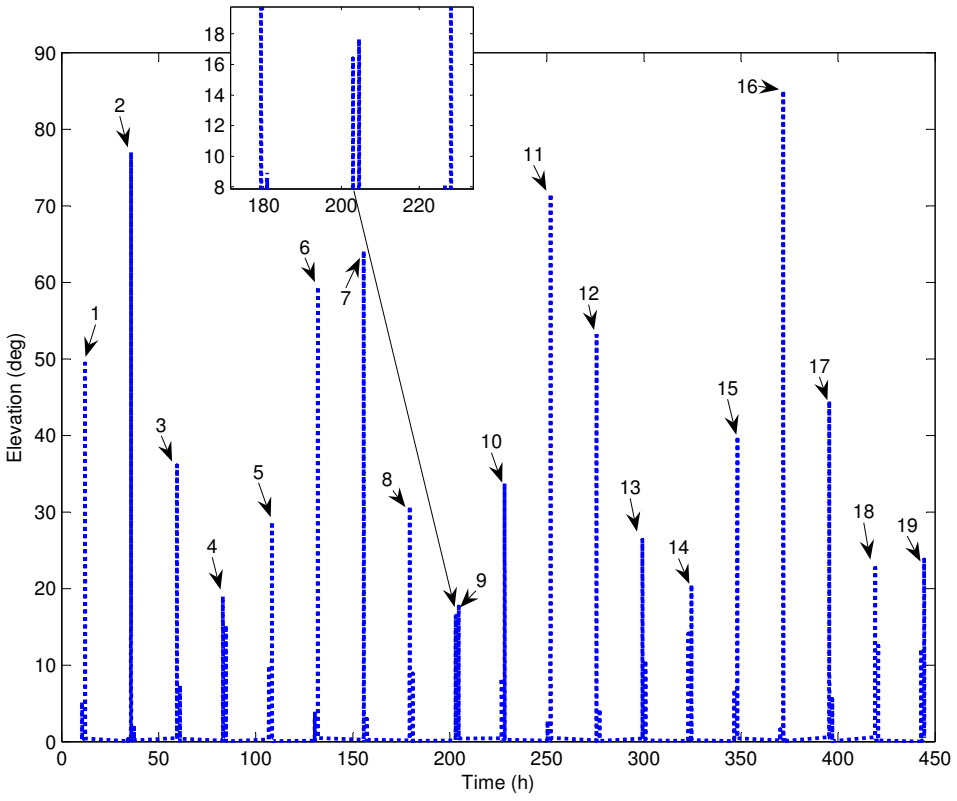


Figure 2.14: Elevation angle calculated for a period of 20 days as the satellite moves from north to south across the ground station.

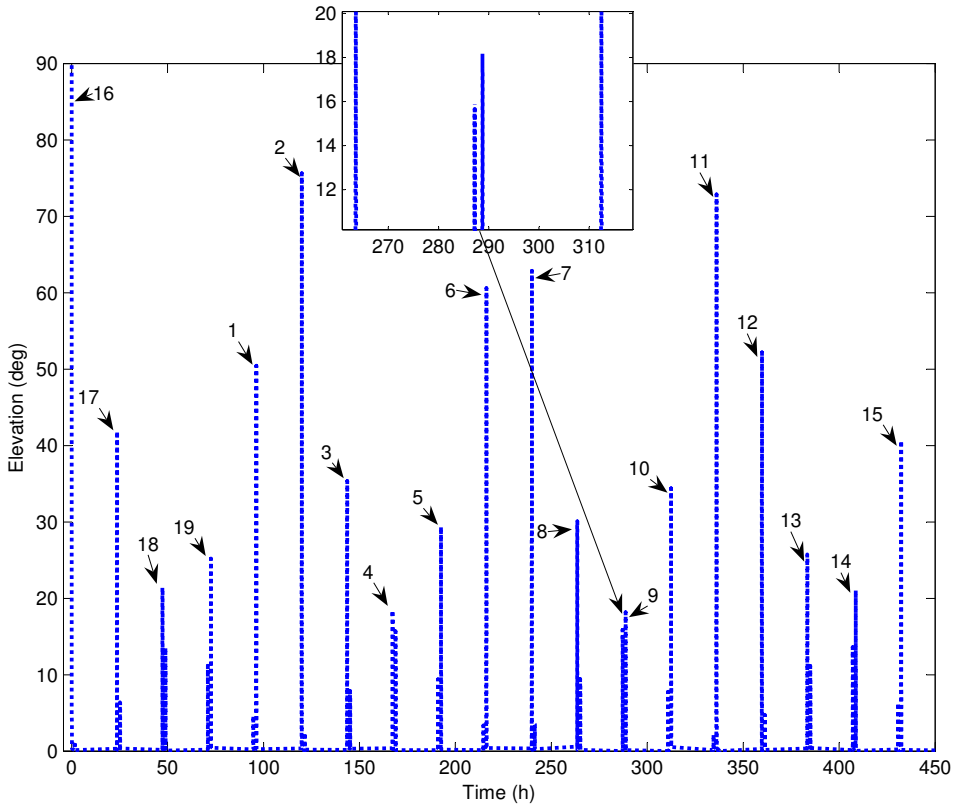


Figure 2.15: Elevation angle calculated for a period of 20 days as the satellite moves from south to north across the ground station.

Figure 2.14 and Figure 2.15 show the elevation angles calculated for both of these two situations, for a period of 19 days. It appears from the two figures that the elevation angles calculated when the satellite moves from North to South across the ground station are shifted by 84 hours for the window calculated when the satellite moves from South to North across the ground station. Thus, the elevation angles in the South-North window are delayed by 84 hours. Therefore one has only to look at one of these windows to get a further indication of the typical elevation angles for a satellite in orbit. The numbering of the elevation angles in these two figures corresponds to this shift in elevation angles. Note, however, that these elevation windows will vary because of the ground station location and the initial satellite coordinates. They only serves to give an indication of the typical elevation angles.

## 2.8 CAN Bus

Controller Area Network (CAN) is a very reliable serial bus system, one reason is having an absence of a host computer. Each device on the network can send and receive data. A CAN node connects such a device to the network.

A CAN node (Figure 2.16) comprises a transceiver, CAN controller and a processor. The transceiver is responsible for the physical interface to the serial CAN bus. The CAN controller implements the CAN protocol and the processor, containing embedded firmware, allows high-level communication functions.

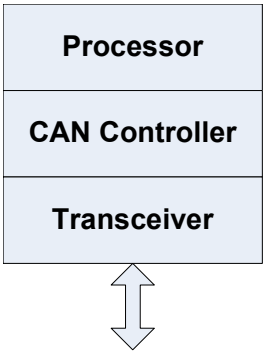


Figure 2.16: CAN node composition

A device usually contains only one CAN node. By creating a virtual CAN node it is possible for a software application on a device to interface to the physical CAN bus. The virtual CAN node uses the hardware of the existing CAN node to communicate on the CAN bus.

The Sunspace CAN bus architecture (Figure 2.17) has two separate CAN busses. One is called the C&DH bus and the other the ADCS bus. It is only possible for a device to communicate with another device on a different CAN bus through the OBC which is connected to both of these bus networks. The OBC has therefore two CAN nodes.

## 2.9 Steerable Antenna Array

This section will discuss the basic architecture of the steerable antenna array (SAA) developed by KU Leuven. Figure 6.16 shows an overview of the SAA. A description of the various components will follow:

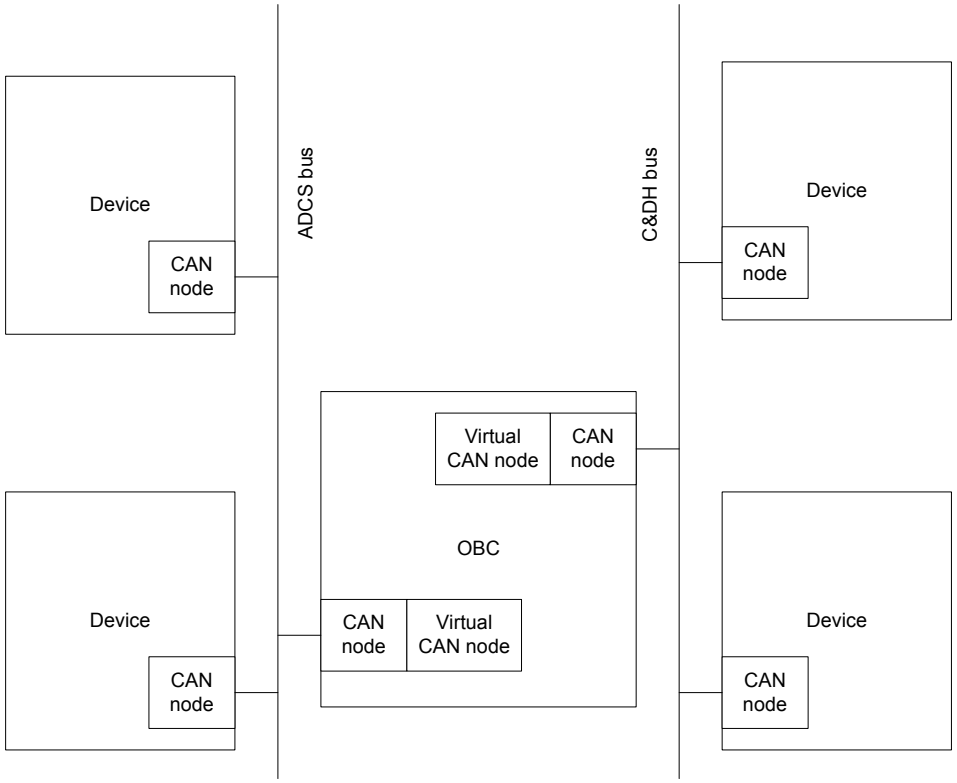


Figure 2.17: CAN bus architecture

**RF RX** The receiver board consists of multiple array elements. These elements convert a received RF signal to its in phase I and quadrature Q component.

**PLL** Provides the reference frequency to the receiver board.

**ADC** Converts the analog I and Q signals to 12-bit digital samples.

**FPGA** Field Programmable Gate Array

**DBS Calculations** Phase shifts the I and Q signals received from the various elements before outputting the summed I and Q signals.

**UART control interface** Provides an interface to control DBS calculations and to set up the PLL. This interface would be accessed by the payload OBC.

The DBS calculations module illustrated in Figure 2.19 consists of a phase shifter and summation components. Each element has a phase shifter, which

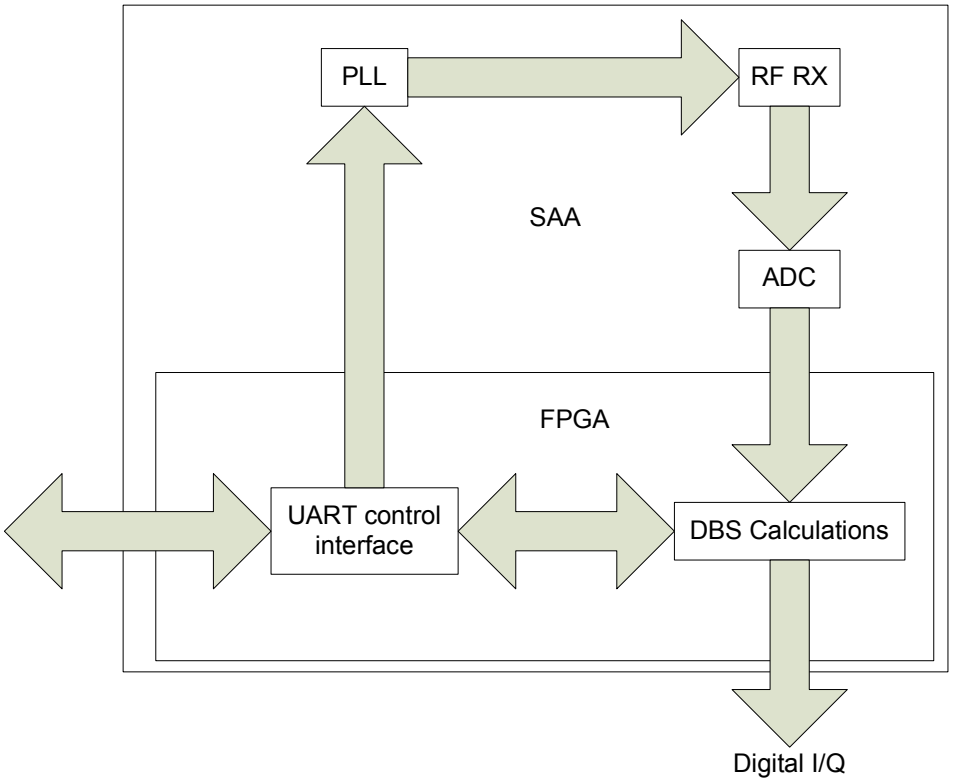


Figure 2.18: Steerable antenna array

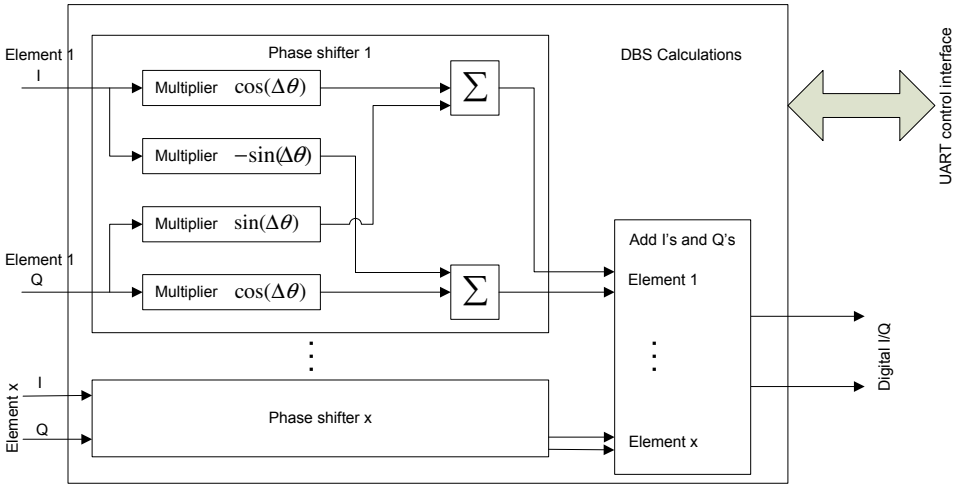


Figure 2.19: DBS calculation component

shifts the received signal. The phase shifter comprises 4 multipliers and 2 summations. The multiplier values are set via the UART control interface

## 2.10 Link budget

A link budget and resulting SNR are used to determine the level and reliability of a communication link assisting with correct component and subsystem selection. The factors affecting the link budget will be discussed in section 2.10.1.

Although this project is intended for aircraft flight, the link budget is calculated for a satellite link. The link budget can be divided into two sections, an uplink and a downlink. Two separate antennas are required on the satellite, one for the uplink and the other for the downlink. The receive antenna is the KU Leuven beam steerable SAA and the transmit antenna is a fixed one set at a different frequency.

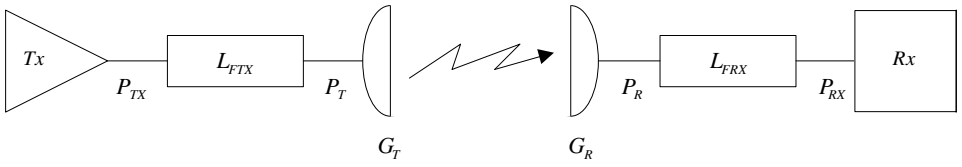


Figure 2.20: Losses in the terminal equipment [13]

### 2.10.1 Factors Affecting the Link budget

#### Carrier Frequency

A S-band link at 2.4 GHz was chosen for this project, this being in the licenced free ISM band. An application for a satellite frequency licence has to be submitted to the International Telecommunication Union (ITU) [14]. This is a rather time consuming process where specific details such as the flight path of the satellite have to be given. These details are not known, at this stage of the project. The ISM band is not guaranteed interference free, and to compensate for this the final flight tests will be done in a less populated, open area.

#### Satellite Receive Antenna Gain

Figure 2.21 shows the radiation pattern of an existing prototype array designed by KU Leuven. Although the prototype was designed at 2 GHz and the

antenna array used in this project operate at 2.4 GHz, the radiation patterns will be similar.

The fact that the antenna array is steerable will greatly improve the link budget, especially for low elevation angles. The coverage or steer angle of the antenna ranges between ±60°. The gain of the antenna will slightly decrease at these high coverage angles. The decrease in gain at these angles was not taken into consideration when the link budget was calculated. The reason for this is that at the time of the link budget system design the antenna array had not yet been developed by KU Leuven and thus the decrease in gain at these angles was not known. It was assumed that the decrease of antenna gain at these angles would be minimal and would thus not affect the link budget severely. It would be possible to construct a more thorough link budget at a later time when these parameters are known. The current calculations are considered adequate at present.

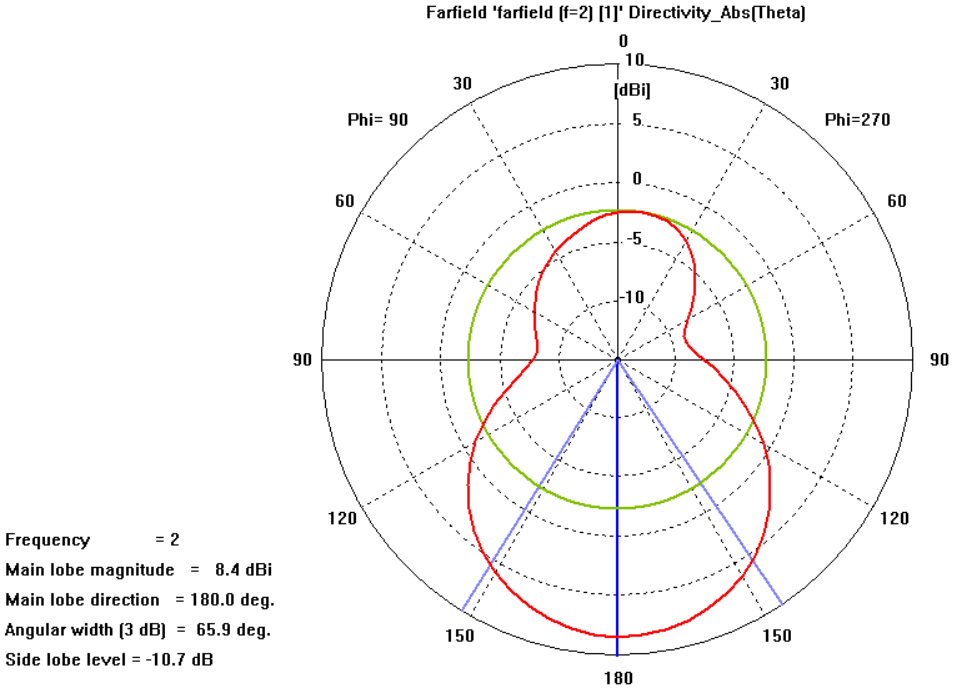


Figure 2.21: Radiation pattern of the prototype array [15]

### Satellite Transmit Antenna Gain

A transmit antenna with a fairly wide radiation pattern will be used on the aircraft. A realistic estimate of 8 dB was made for the antenna gain.

### Ground Station Antenna Gain

A quad helix antenna will be designed for use as the ground station antenna. This antenna has a doughnut shape, or if a cross section is taken, a heart shaped radiation pattern. This means that this type of antenna has greater gain at lower elevation angles, which is ideal for the a ground station antenna, because it is at lower elevation angles that the most $L_{FS}$ losses occur. As shown in section 2.7, it is also worth noting that the satellite will pass a ground station most often at these lower elevation angles. Figure 2.22 shows the desired radiation pattern for the quad helix antenna.

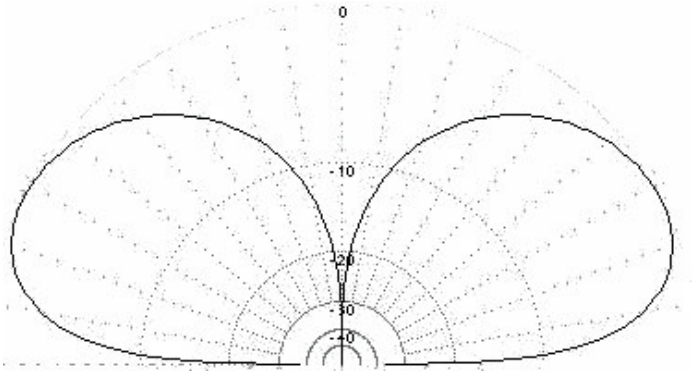


Figure 2.22: Radiation pattern of a quad helix antenna [16]

### Free Space Loss

Free space losses are due to the distance between the transmitter and receiver, which is the main contributor to losses in the transmission link.

$$L_{FS} = \left(\frac{4\pi D f}{c}\right)^2 \tag{2.10.1}$$

where $D$ is the distance between the satellite and ground station calculated in section 2.6.5, $f$ the carrier frequency and $c$ the constant for the speed of light.

### Atmospheric Loss

At 2.4 GHz atmospheric effects such as rain, clouds, snow and ice do not have a significant influence on the link budget and can be ignored.

### Feeder Loss

The losses between the transmitter and antenna denoted $L_{FTX}$ are very small, as are the $L_{FRX}$ feeder losses between the antenna and the receiver. An estimate of 0.5 dB was made for these losses.

### Polarisation Mismatch Loss

As suggested in [13], polarisation mismatch losses occur when the orientation of the polarisation of the receiving antenna differs from that of the received signal. Atmospheric effects experienced can change the polarisation of a signal. A realistic approximation for the polarisation mismatch loss of 3 dB was made, for this initial phase of the project.

### Noise Floor

The noise floor is constituted by antenna and receiver noise temperatures. The noise is a function of temperature and the bandwitdh.

$$P_{NoiseFloor} = k_B T_{tot} B \tag{2.10.2}$$

where
Boltzmann's constant $k_B = 1.38 \times 10^{-23}$
Total noise temperature $T_{tot}$
Bandwitdh of bandpass filter $B$

The noise temperature $T_{tot}$ can be calculated by

$$T_{tot} = T_{ant} + T_{rec} \tag{2.10.3}$$

where $T_{ant}$ is the noise temperature of the antenna and

$$T_{rec} = (F-1)T_0 \tag{2.10.4}$$

converts the noise figure $F$ of the receiver to the noise temperature of the receiver. $T_0 = 290\,\mathrm{K}$ is the reference temperature.

### Doppler Effect

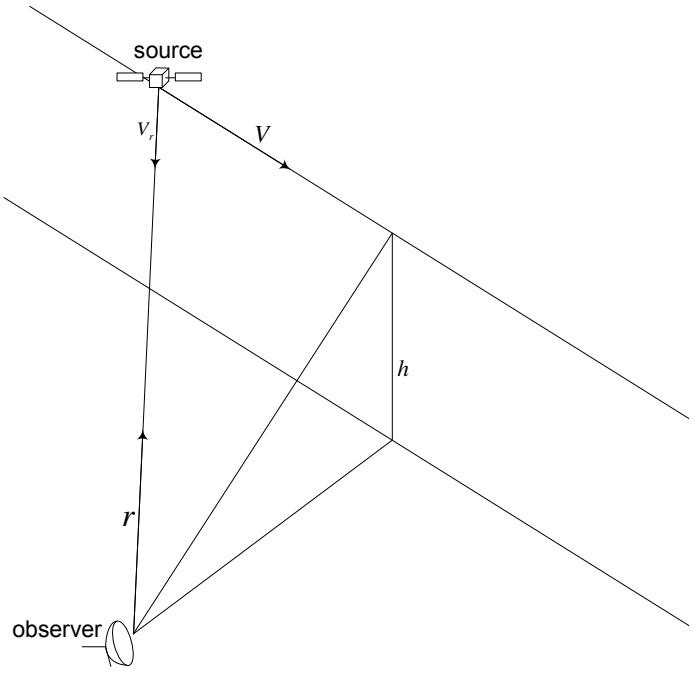


Figure 2.23: Illustrates the vectors used to calculate a frequency shift for the downlink.

The frequency of a propagating wave changes relative to an observer, if either the observer or the source of the wave move relative to each other. The carrier frequency of the transmission link between the satellite and ground station shifts, as the satellite orbits the earth. The frequency will increase if the satellite moves towards the ground station, but will decrease if the satellite moves away from the ground station. The frequency shift can be calculated as follows.

$$\triangle f \quad = \quad V_r \frac{f}{c} \tag{2.10.5}$$

where $c$ is the velocity of light, $f$ is the frequency of the transmitted signal and $V_r$ is the relative velocity component between the source and the observer.

The relative velocity $V$ is the velocity of the source relative to the observer. $V_r$ is the component of $V$ along the line connecting the source and observer. Figure 2.23 illustrates this relationship. By expressing the satellite velocity as

a vector $\vec{v}$ and with $\vec{r}$, a position vector pointing from the observer towards the source, reduces Equation 2.10.5 to.[17]

$$\triangle f = -\frac{\vec{v} \cdot \vec{r}}{|\vec{r}|} \frac{f}{c} \tag{2.10.6}$$

### 2.10.2 Calculations

See Section 4.8.1 on page 66 for the link budget calculations and Appendix A for the results of the calculations.

## 2.11 Conclusion

A description of the concepts needed to develop the system was presented in this chapter, in order to enable better understanding of decisions set out in the subsequent chapters. The first of these chapters discuss the emulation strategy and is presented next.

# Chapter 3

# Emulation Strategy

This section present a few emulation strategies as considered and the reasons for selecting a particular one.

Two main emulation approaches were considered. The first approach was to emulate the satellite position relative to a ground station and the second was to emulate the ground station position relative to a satellite. It is clear that for the first case the position could be described by elevation and azimuth angles [18]. The second approach describes the position of a ground station from the perspective of a satellite using the $\phi$ and $\theta$ angles. Figure 3.1 shows the defined angles. These angles are, in both cases, time dependant for LEO satellites. The function of the emulator is to calculate a flight route for an aircraft that would approximate these orbital characteristics as closely as possible.

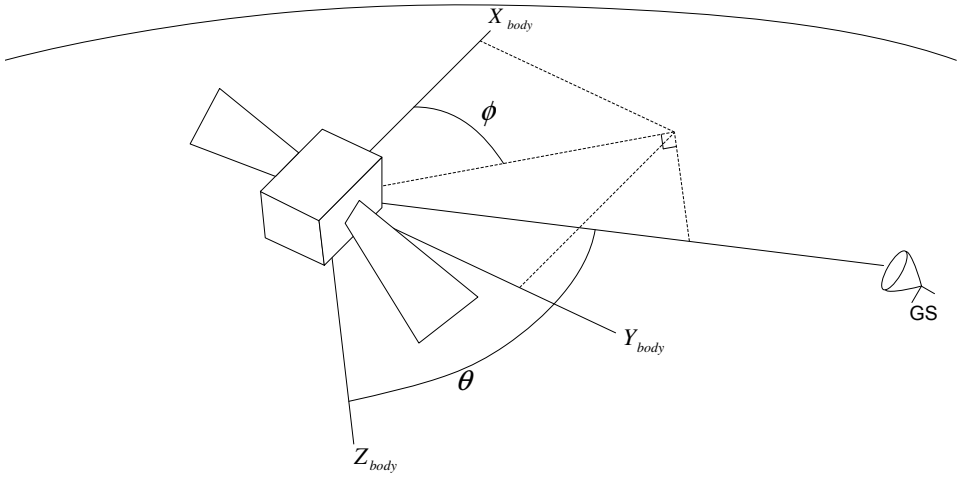


Figure 3.1: $\phi$ and $\theta$ angle definitions

## 3.1 Flight Strategy

1. The first emulation flight strategy is to fly in ascending concentric circles around a ground station. This strategy covers all the azimuth or $\phi$ angles for a specific elevation or $\theta$ angle. By spiralling upwards it is possible to cover many elevation or $\theta$ angles. The implementation of this strategy is however arduous. It is difficult for an aircraft to fly at a constant speed in an accurate, circular, upwards path around a ground station. However, the main disadvantage of this option is that the specific time variant behaviour of practical elevation-azimuth or $\phi$-$\theta$ angles are not taken into account.

2. The second strategy is to fly past a ground station in a straight path parallel to the earth surface at a constant speed and altitude. It is easier for a pilot to implement this strategy than the previous one and he will be able to maintain a more stable attitude. Because of this and with the aircraft flying parallel to the surface, the orientation of the antenna on the aircraft will match the predicted orientation of the antenna on the satellite more closely. The orientation of the antenna will enable the steering angles of the antenna to approach that of the actual satellite implementation, providing a more realistic scenario. The orientation will also facilitate the calculation of a more accurate linkbudget for a flight path. The linkbudget can then be emulated by compensating for the $L_{FS}$ losses by attenuating the transmitting or receiving signal. The further advantage of this strategy is that the specific time variant behaviour of the elevation-azimuth and $\phi$-$\theta$ angles of a LEO satellite are taken into account. This will also enable the relationship between the antenna steering angles and time to match that of the satellite application. For these reasons the second strategy is clearly the better one and was selected for actual implementation.

## 3.2 Transmission Link Strategy

With the aircraft based flight test, the direct LOS distance to a ground station, is clearly much shorter than in the case of a real satellite. In order to emulate the satellite link budget, the free space loss (FSL) must be compensated for.

The aircraft link must, therefore, be attenuated to achieve the free space loss of a satellite link.This could be simply effected by adjustment of the transmit power for both the up- and downlinks. Doppler shift is not a consideration for the aircraft flight test due to the low speed, but certainly is for the satellite link. The amount of required compensation will be determined by final orbit and receiver front-end selectivity bandwidth. For this project, Doppler compensation will probably be performed at the ground station and therefore, no compensation has been implemented on the emulator platform. To minimise the affect of terrain scattering, the emulation flight tests are planned for a wide open semi-desert area.

## 3.3 Calculation of Aircraft Parameters

In order to implement the chosen strategy as discussed in the previous section, it is necessary that the aircraft flight parameters be calculated in terms of the required trajectory. This calculation was done by means of a suitable script, based on the elevation-azimuth approach, as explained in Section 3.1.

The script is fed with the maximum elevation angle as an input parameter. The maximum elevation angle occurs when the object is closest to the ground station. The script then calculates the time values for a LEO satellite in orbit, as it transits from minimum- to maximum elevation angle. An iterative method is implemented to calculate the parameters for the aircraft flight path, emulating the satellite elevation time window. An elevation time window is calculated for each combination of aircraft altitude and speed. It should be noted that if the altitude changes for a specific elevation angle, so does the minimum distance to the ground station, which is the distance from the ground station to the satellite nadir point, when the aircraft is closest to the ground station.

Figure 3.2 compares the satellite and aircraft elevation angles versus time, for a chosen flight path. It is clear from Figure 3.2 that only a small interval of the visibility time period of an aircraft is suitable to emulate the behaviour of the time varying elevation angles of a satellite. It is for this reason that the graph of the aircraft flight path is shifted to the left, to align the maximum elevation angles. The optimisation of the elevation time graph is achieved by calculating the area under each graph for a specified time window and sub-

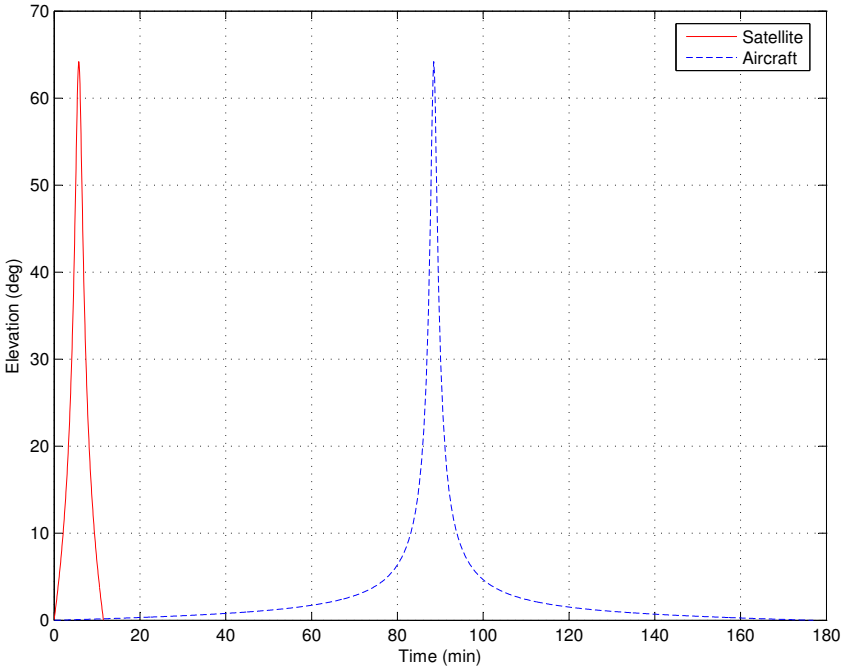


Figure 3.2: Elevation angle versus time

tracting the results. The smaller the result after the subtraction, the better the match between the two graphs. The results of these calculations are obtained from the script in the form of two figures. The dark blue areas in Figure 3.3 specify the areas where the speed and altitude of the aircraft best conform to the emulated elevation and azimuth angles. Figure 3.4 shows the distance for different altitudes from the aircraft's nadir point to the ground station at the point when the aircraft is closest to the ground station. Figures 3.3 and 3.4 enable us to choose either the desired speed, altitude or distance from the ground station and then use the figures to calculate the other parameters. Therefore, using the results from Figures 3.3 and 3.4, will allow us to specify the final flight route as required.

It is thus possible to calculate aircraft flight path parameters of speed, altitude and distance from the ground station to satisfy the elevation and azimuth angles as related to the satellite's orbital flight. Figure 3.5 and Figure 3.6 illustrate this conformity between the elevation and azimuth angles of the satellite and the aircraft for the visibility time period of the satellite. Although deviations will occur at low elevation angles, a very useful time window for testing purposes can still be obtained.

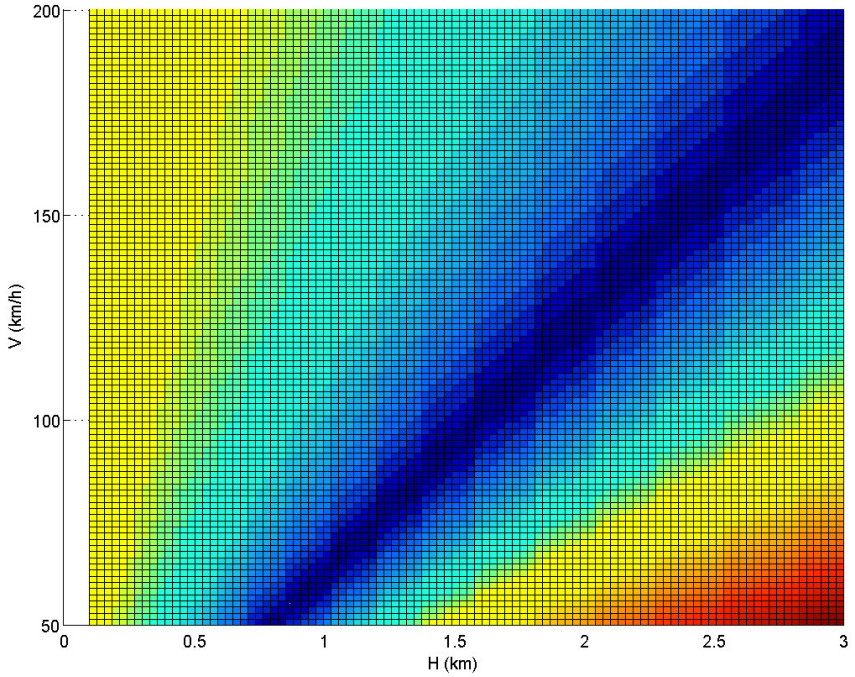


Figure 3.3: Speed versus altitude of aircraft

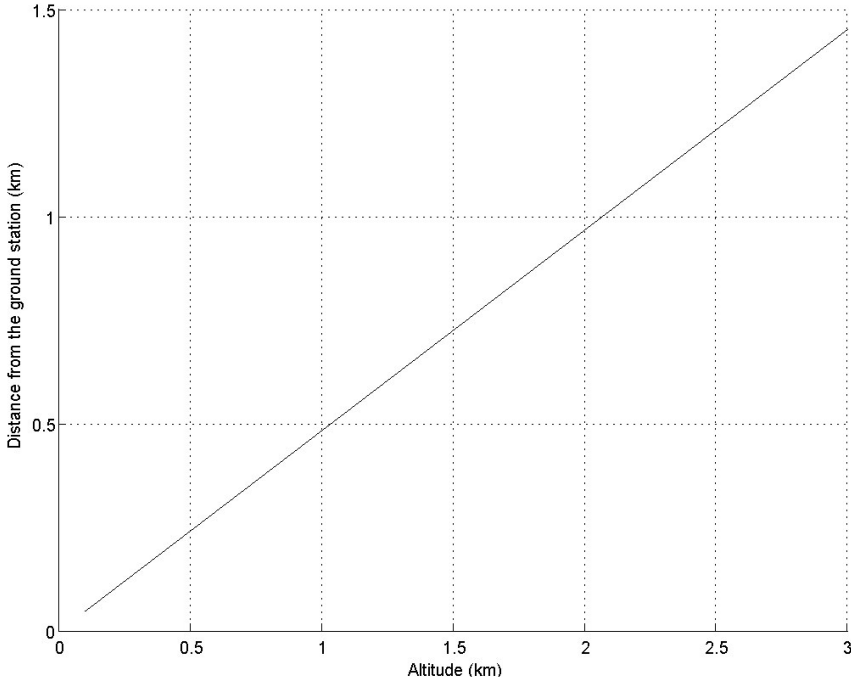


Figure 3.4: Distance from ground station

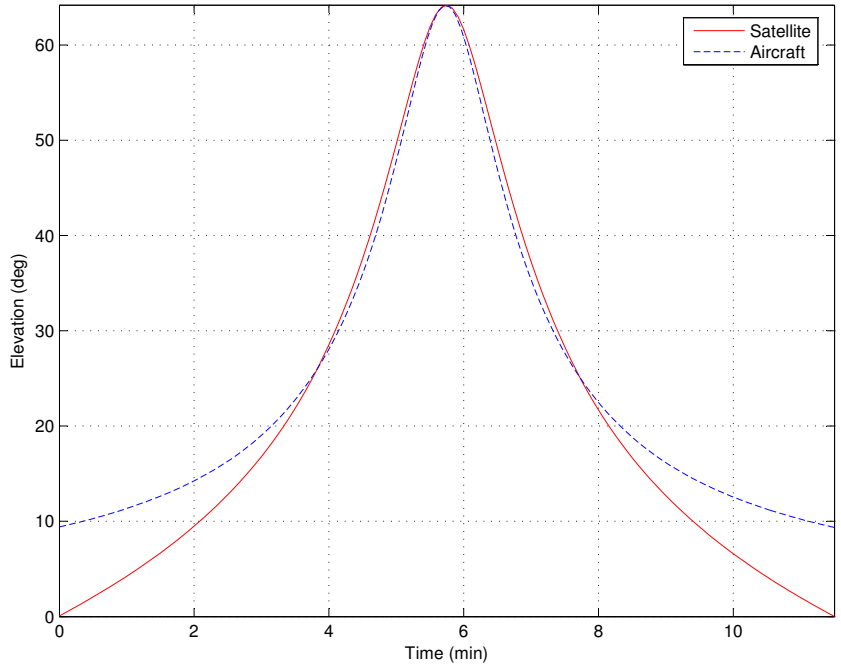


Figure 3.5: Elevation angle versus time interval

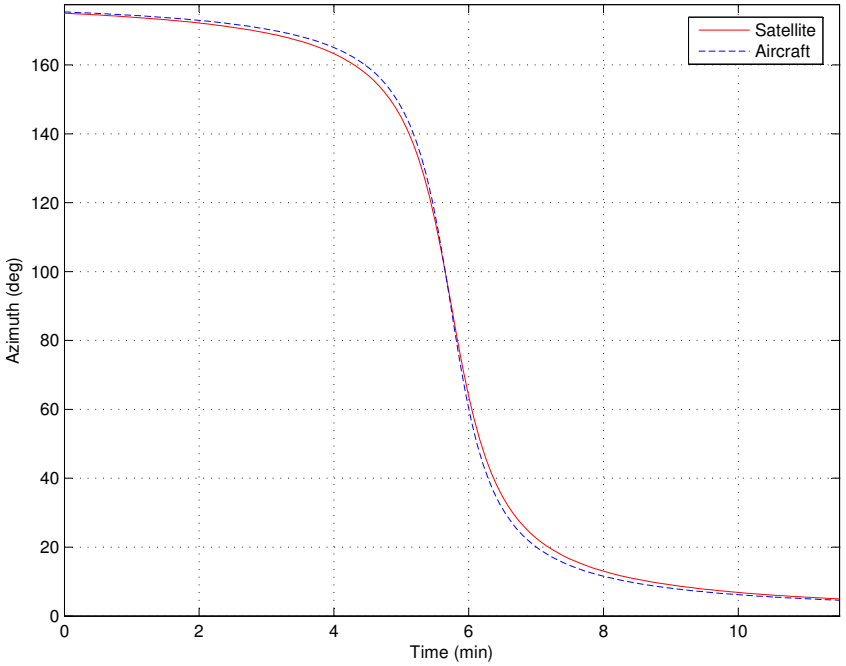


Figure 3.6: Azimuth angle versus time interval

## 3.4 Conclusion

This chapter presented an emulation strategy, that constituted a flight and transmission link strategy. The main concept of the flight strategy is to fly with constant speed and altitude past a ground station, emulating a LEO satellite's elevation and azimuth angles. The transmission link is then attenuated to compensate for the lower free space losses. The aircraft parameters that would enable such a flight profile, were all calculated and presented.

The next chapter will discuss the system architecture to be used in conjunction with the emulation strategy enabling the overall system to closely emulate a LEO satellite platform.

# Chapter 4

# Systems Design

This chapter will provide a description of the systems design. Decisions as set out in this chapter will affect later implementation of the system.

## 4.1 System Architecture

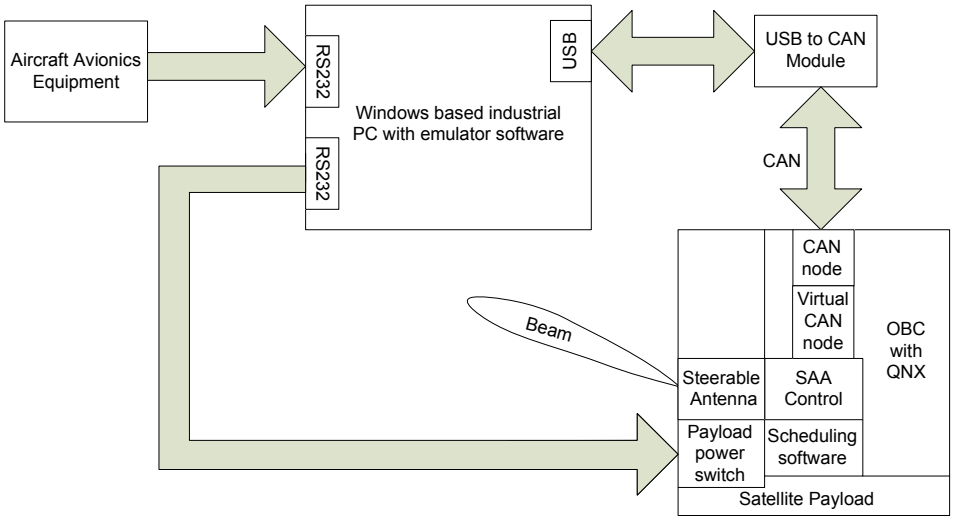


Figure 4.1: System diagram

The system design comprises two sections, the emulator and the satellite payload modules. The components indicated in the system diagram, Figure 4.1 have a direct influence on the system design. Components that do not directly affect the system are not shown here.

### 4.1.1 Satellite Payload Module

The following components on the payload are applicable to the system design:

- Steerable antenna developed by KU Leuven.
- The OBC, an SH4 processor with a 32-bit RISC architecture.
- The QNX operating system runs on the OBC. This real-time operating system is UNIX based which was provided by SunSpace with additional software components.
- The scheduling module is software that schedules the communication times between the satellite and various ground stations.
- A steerable Antenna Array (SAA) control module electronically controls the steerable antenna array. The required control algorithm is developed in this thesis. The SAA module is housed in the OBC of the payload.
- The CAN node, which allows devices to connect to the CAN network.
- A virtual CAN node enables the SAA control software to use the hardware of the exsisting CAN node on the OBC to communicate over the CAN bus.

### 4.1.2 Emulator Module

Instead of connecting the payload to a satellite through the CAN bus, the payload is connected to the emulator module, which mimics the behaviour of an actual satellite. The emulator module comprises an industrial PC with emulator software, known as the Aircraft Satellite Emulator (ASE), which provides an interface for the user to construct a flight path for an airborne object that emulates the orbital characteristics of a satellite pass as closely as possible. As the emulation strategy is to fly the payload with the emulator module on an aircraft, the emulator module connects to the aircraft avionics equipment. Just as the satellite would provide data to the payload in flight, the emulator will provide the necessary data.

## 4.2 Functional analysis

A functional analysis of the system is presented in this section. By defining various hardware and software modules, a better understanding of the system can be achieved. Figure 4.2 shows the system's functional block diagram.

### 4.2.1 Functional Units

**User interface (FU1)** Allows the user to set up and view a scenario.

**Scenario (FU2)** Stores the scenario set up by the user.

**Calculation engine(FU3)** Uses the parameters provided by the scenario to calculate the appropriate altitude and minimum distance from the ground station that the aircraft has to fly to emulate a satellite.

**Process data (FU4)** Stores the calculated data.

**Control unit (FU5)** Controls the functional blocks and data flow of the emulator module on the PC.

**Flight calculation engine (FU6)** Calculates various parameters in flight. For example the aircraft elevation and azimuth angle.

**Comms interface (FU7)** Provides a communication interface to the following modules: Firstly to the aircraft avionics equipment on board the aircraft and secondly to the payload. The data sent by the aircraft avionics equipment is sent serially in a specific packet format. The task of the comms interface is first to recognise a valid data packet and then to decode the received data packet. The second communication interface is to the payload. More specifically, to the payload power switch. This switch is basically a relay that switches the power of the whole payload on or off. The payload will therefore be switched on or off by the comms interface toggling the voltage on the serial port high or low.

**Aircraft avionics (FU8)** This functional block represents the aircraft avionics equipment on board the aircraft. The avionics equipment sends data packets serially, describing the coordinates and attitude of the aircraft.

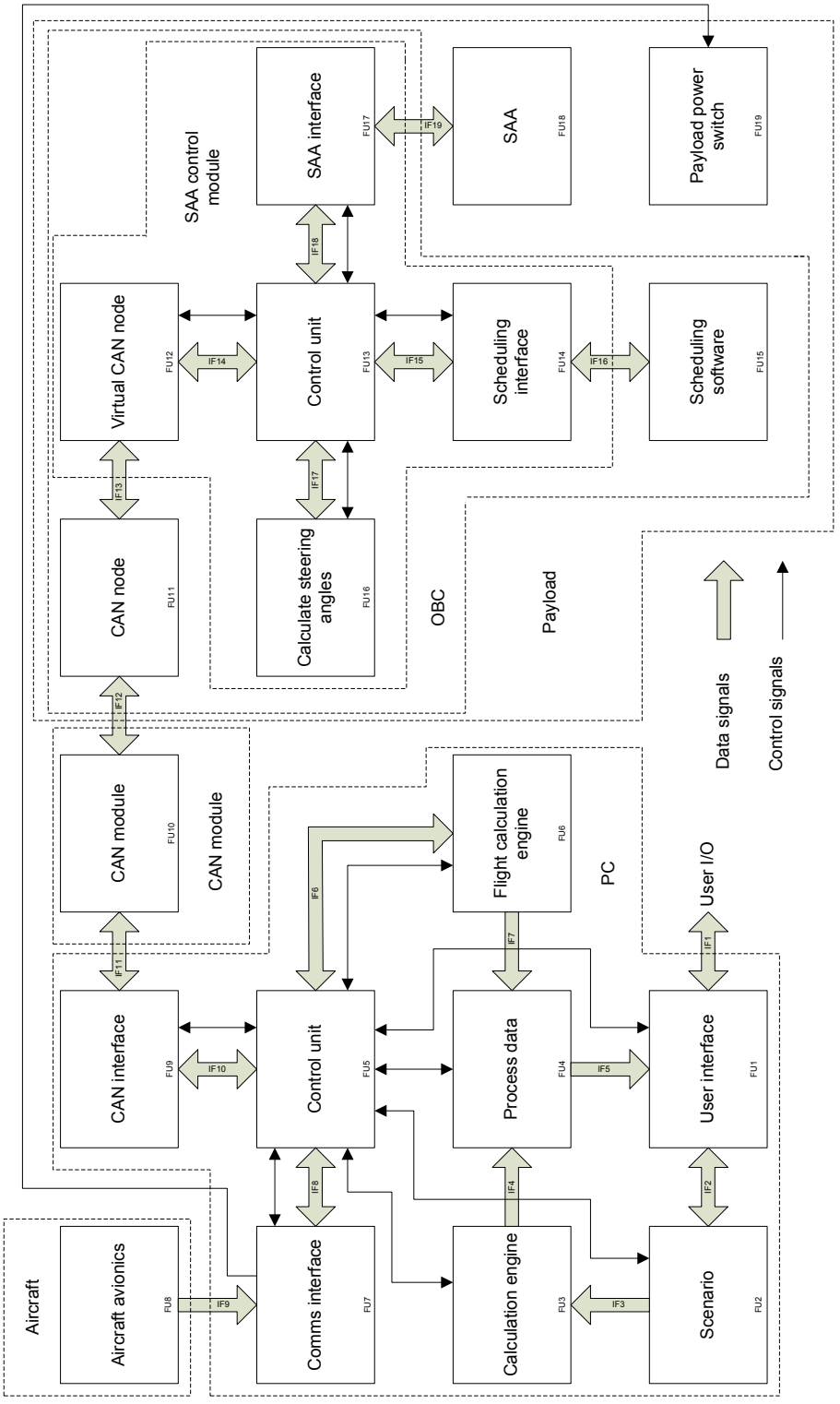


Figure 4.2: Functional block diagram of the system

**CAN interface (FU9)** This CAN interface is on board the PC. The task of this interface is to communicate over the CAN bus. This is achieved by controlling the CAN module (FU10) through the USB port. The CAN module is controlled by using software routines supplied by the manufacturers of the CAN module.

**CAN module (FU10)** The CAN module creates a USB interface enabling a PC and CAN bus to connect. By using the routines and device drivers supplied by the CAN module it is possible to link the aircraft satellite emulator (ASE) on the PC to the CAN bus. The PCAN-USB module from PEAK-System was selected to implement this CAN module.

**CAN node (FU11)** This CAN node allows the OBC on the payload to connect to the CAN bus.

**Virtual CAN node (FU12)** This virtual CAN node allows software on the OBC to use the existing hardware of the OBC CAN node to communicate over the CAN bus.

**Control unit (FU13)** Controls the functional blocks and data flow of the SAA control module on the OBC of the payload.

**Scheduling interface (FU14)** The scheduling interface has the task of obtaining the coordinates of the ground station, toward which the antenna should be directed, from the scheduling software (FU15). The scheduling interface requests this information from the scheduling software and will then wait for a reply that contains the requested data. After the requested data is received, the scheduling interface sends another request to the scheduling software and repeats this process.

**Scheduling software (FU15)** The scheduling software creates a schedule which assigns communication times with the satellite for various ground stations. Requests received from the SAA control module are handled by sending a reply containing the requested data. The data sent are the coordinates of the target ground station. Note, however, that the reply is sent only at the appropriate time, when the antenna should be directed from its previous target position to a new target ground station position. This implies that if a request is received the reply will be sent only if the antenna should be directed to another ground station.

**Calculate steering angles (FU16)** This functional block calculates the defined theta and phi steering angles that are necessary to direct the antenna beam towards a specific ground station.

**SAA interface (FU17)** The SAA control module connects to the SAA through a serial port. By sending data packets, in a specific format, serially to the SAA, the SAA control module can control the SAA.

**SAA (FU18)** The steerable antenna array developed by KU Leuven.

**Payload power switch (FU19)** The payload power switch switches the power to the whole payload on or off. The module contains a relay that can be switched by a serial port. Thus the payload is switched on or off, depending whether the voltage on the serial port is high or low.

### 4.2.2 Interfaces

**IF1** A keyboard and mouse provides user input to the system. A screen displays the scenario visually to the user.

**IF2** Scenario parameters are read from or written to the user interface.

**IF3** The scenario parameters are transferred to the calculation engine (FU3).

**IF4** Data calculated in the calculation engine (FU3) are sent to the process data functional block (FU4).

**IF5** Data stored by the process data functional block (FU4) are sent to the user interface (FU1) in order to display the data to the user.

**IF6** Flight data received by the control unit (FU5) are sent to the flight calculation engine (FU6). Some results of the calculations are returned to the control unit (FU5). These results indicate whether the control unit should switch the payload on or off.

**IF7** The results of the calculations are sent from the flight calculation engine (FU6) to the process data functional block (FU4).

**IF8** Decoded aircraft avionics equipment data are sent to the control unit (FU5). A data message that controls the power status of the payload is sent by the control unit (FU5) to the comms interface (FU7).

**IF9** Aircraft avionics equipment data are transmitted serially to the comms interface (FU7).

**IF10** Decoded data received from the CAN bus and data to be transmitted on the CAN bus are transferred over this interface.

**IF11** This interface transfers CAN packets between the CAN interface (FU9) and the CAN module (FU10) through an USB port.

**IF12** CAN packets are transmitted on the CAN bus.

**IF13** This interface transfers CAN packets between the CAN node (FU11) and the virtual CAN node (FU12). CAN packets addressed to this specific CAN node (FU11) are let through to the virtual CAN node (FU12). CAN packets that must be transmitted on the CAN bus are sent from the virtual CAN node (FU12) to the CAN node (FU11).

**IF14** Data received from the CAN bus and data to be sent on the CAN bus are transferred by this interface.

**IF15** An enquiry message is sent by the control unit (FU13) asking the scheduling interface if there are new target coordinates. If there are, the scheduling interface (FU14) replies with the new coordinates of the target ground station.

**IF16** Requests for target coordinates and the reply containing the coordinates are transmitted over this interface.

**IF17** The data necessary to calculate the steering angles are sent to functional unit FU16. The results, containing the steering angles, are then transmitted back to the control unit (FU13).

**IF18** The antenna steering angles are sent by the control unit (FU13) to the SAA (FU18) interface.

**IF19** Commands directing the antenna beam are sent serially over this interface to the SAA (FU18).

## 4.3 System-Wide Design Decisions

The following initial design decisions were made at the start of the project:

**Emulator programming language and development environment** Borland's C++ Builder 6 [19] was chosen as the development environment for the emulator software. The environment uses C++ as programming language and provides various tools, such as drag and drop buttons, that allow the user to create a GUI. Prior experience with this development environment and facilitation of faster development influenced the selection of this development environment. However, the disadvantage of this environment is that the developed software can run only on Windows. But, because the portability of the emulator software is not crucial (it need only run on the emulator PC), it was found acceptable to choose Windows as the operating system.

**Emulator hardware** An industrial PC was chosen to implement the emulator software, being relatively small and robust. These qualities are especially important when it is considered that the emulator hardware will be mounted in an aircraft with limited space. The robustness of the system is crucial in this environment. The main reason this PC is so robust is that compact flash is used instead of a solid state hard drive. Although the capacity of the compact flash used in this PC is smaller than that of the typical solid state hard drive, was it found to have ample space to run an operating system and the emulator software. The PC used for the emulator module has the following specifications:

**Emulator computer** Portwell WADE-1120 Industrial PC

**CPU** Intel Celeron M 600MHz

**Storage** 2GB CompactFlash

**Emulator operating system** Windows XP was chosen as the operating system for reasons already mentioned.

**USB to CAN module** A USB to CAN module was chosen to connect the PC to the CAN bus. The PCAN-USB module from PEAK-System was selected to fulfil this purpose.

**Aircraft** A light aircraft is necessary to implement the flight strategy. The Jora [20], which is a light aircraft that belongs to Stellenbosch University, was selected to perform the flight segment of this project. This aircraft is available to the engineering faculty for various projects and consent was given to use this aircraft for the project. See Figure 4.3 for a photo.

**CAN protocol** The CAN protocol must conform to the standards of local satellite constructor SunSpace.

**SAA Control module programming language** The programming language C was chosen to develop the SAA Control module software. This language was already in use to develop software components for the SH4 that runs on QNX, prior to this project. No reason was seen to deviate from this choice.

Figure 4.3: Jora

## 4.4 Concept of Execution

UML use case and sequence diagrams are used to model the system's execution. The systems that interact with this system, are represented in these diagrams as actors.

### 4.4.1 Use Cases

All the actors that interact with this system and their corresponding use cases are shown in Figure 4.4. The emulation scenario is set up in the first use case. This is done by the user creating a flight route. A map with the flight route scenario is then displayed to the user. The second use case demonstrates the peripheral devices of the emulator being initialised by the user. These peripheral devices include the interface of ASE with the aircraft avionics equipment and the payload. Data is received by the system from the aircraft avionics equipment in use case three. The data received describes the attitude, as well as the position, of the aircraft. The system switches the payload "on" when the aircraft reaches the start of the flight route and "off" when the aircraft reaches the end of the flight route. This is illustrated by use case four, which simulates the situations as the satellite switches the payload on when communication with a ground station should start and off when communication ends. In use case five, QNX starts the execution of the antenna control software onboard the SH4. This will occur each time the payload is switched on and the SH4 boots. The scheduling software sends the coordinates of the target ground station to the system in use case six. The steering angles of the antenna are then calculated and the corresponding commands sent to the steerable antenna.

### 4.4.2 Sequencing

The sequential interaction of the various components defined by the use cases are illustrated by the sequence diagram of Figure 4.5. Note the following:

- The ASE continuously receives aircraft data from the aircraft avionics equipment.
- The ASE switches the payload on once the aircraft is at the start of the flight path. By switching the payload on the steerable antenna will activate and QNX on the SH4 will boot. QNX will then start the execution of the scheduling and SAA control software. However, note that when the aircraft reaches the end of its flight path the ASE will switch the payload off. This will power down the steerable antenna, as well as QNX and the applications that runs on it.

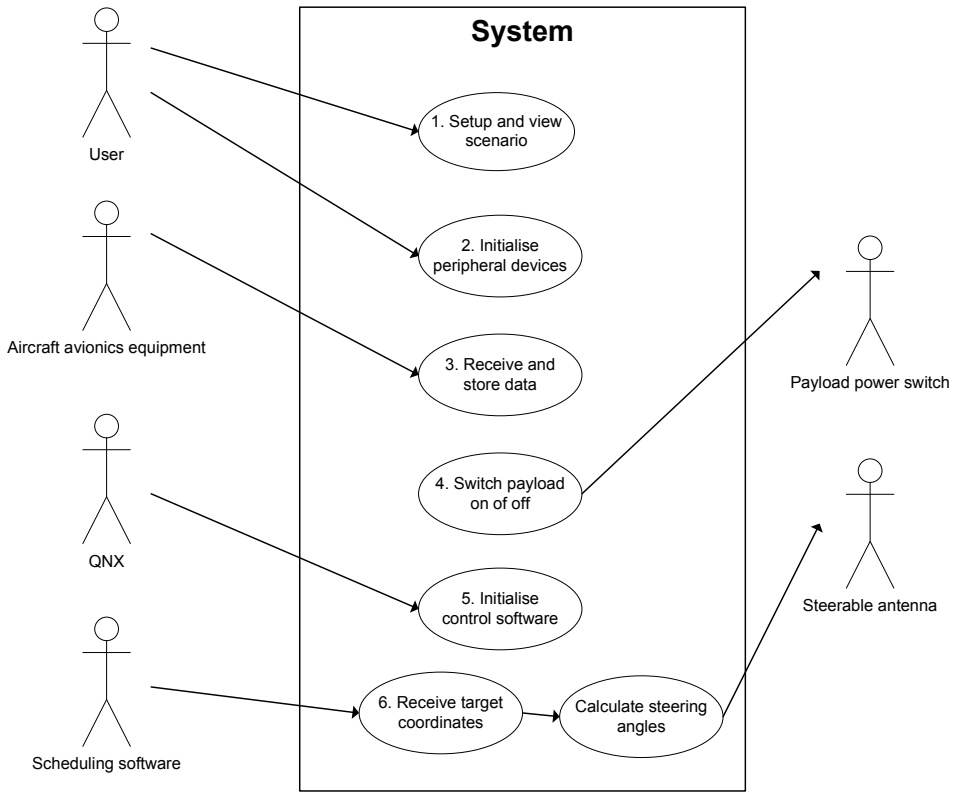


Figure 4.4: System use case diagram

## 4.5 Aircraft Satellite Emulator

The function of the Aircraft Satellite Emulator (ASE) is to emulate the behaviour of a real satellite and its subsystems for the particular payload, while the entire system is flown in an aircraft. For the payload it would be no different from being connected to the network bus of an actual satellite. For this particular development, a Controller Area Network (CAN) bus conforming to the standards of local satellite manufacturer SunSpace is used. The payload OBC can access the telemetry and output information of the emulated subsystems via CAN bus telemetry requests. The emulation software was implemented by means of C++ Builder. The ASE is connected to the aircraft avionics equipment via a RS232 port and to the payload CAN bus via an USB to CAN module.

The aircraft satellite emulator design comprises three sections, ie the aircraft avionics equipment interface, the ASE-payload interface and the satellite emulator software running on the industrial PC. Some of the ASE functional

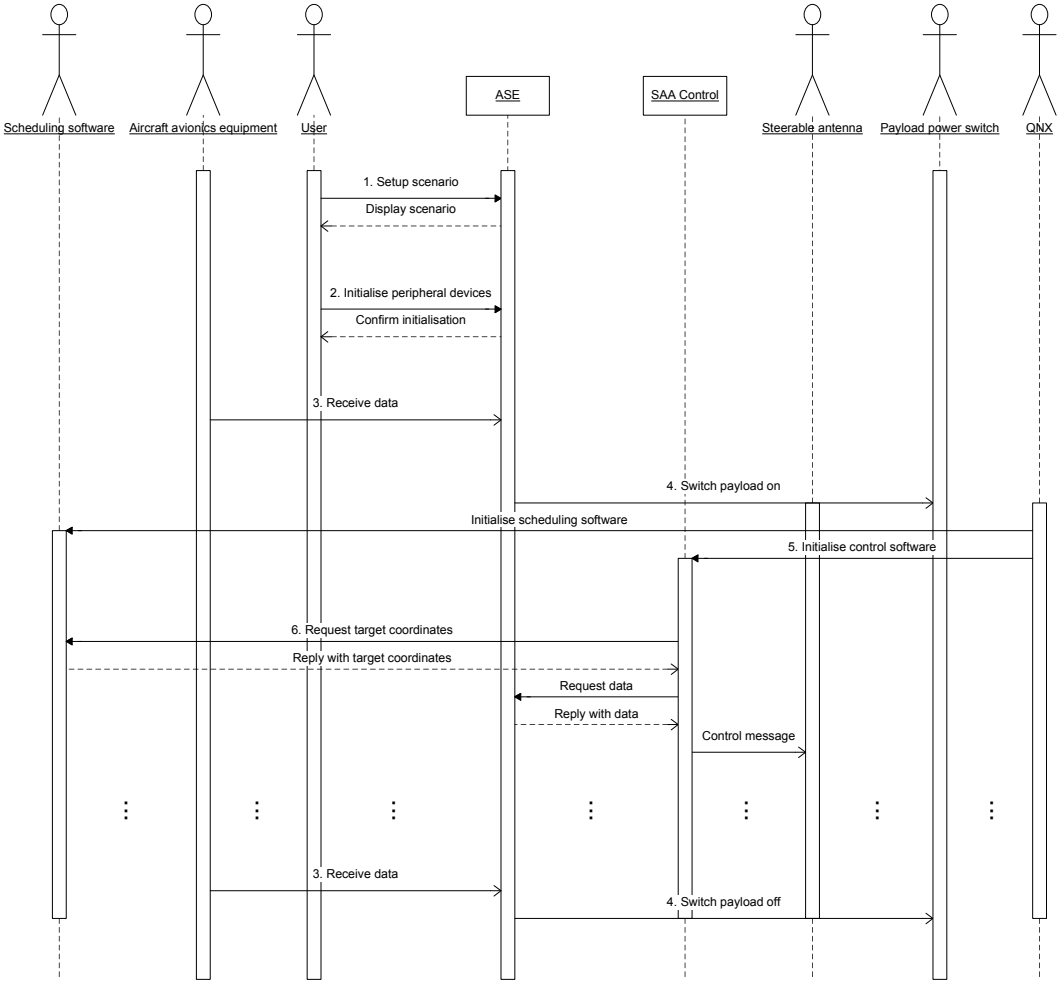


Figure 4.5: System sequence diagram

units presented will be discussed in more detail in the following subsections.

### 4.5.1 ASE-Payload Interface

The PCAN-USB device creates a USB interface that enables a PC and a CAN bus to connect. By using the routines and the device drivers supplied by the PCAN-USB module it is possible to link the aircraft satellite emulator (ASE) to the CAN bus. With the help of these functions it is thus possible to create a CAN bus driver.

The driver can be implemented by a thread that constantly monitors the CAN bus. Each time a CAN message is received, the driver decodes the CAN message by assessing the message identifier. The CAN driver will reply with

the requested data if a valid CAN message addressed to the aircraft satellite emulator is received. Figure 4.6 shows a activity diagram of the CAN driver.

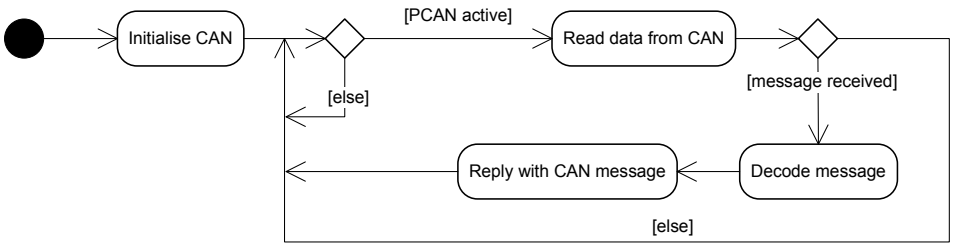


Figure 4.6: CAN driver activity diagram

### 4.5.2 Aircraft Avionics Interface

Various telemetry packets are transmitted serially by the aircraft avionics equipment via the COM port. See [21] for a more detailed description of these telemetry packets.

The aircraft avionics equipment driver (Figure 4.7) is responsible for reading the telemetry data sent serially by the aircraft avionics equipment. The driver is implemented by a thread that continuously monitors the COM port. If telemetry data is transmitted, the driver reads the data from the COM port. The data read is then stored in a buffer. The data packet must then be decoded according to the protocol to verify that a valid message was received. If a valid message was received, the message is decoded and the telemetry data contained in the message stored. The specific telemetry packet that this driver will read, contains the aircraft latitude, longitude, altitude, roll, pitch and yaw.

#### Protocol

The aircraft uses a specific format to transmit data packets. The protocol ensures more reliable data and enables us to recognise specific data. The string layout protocol described by [22] is implemented on the aircraft. The format of the data packets for this protocol is summarised below.

$ II Data * CS

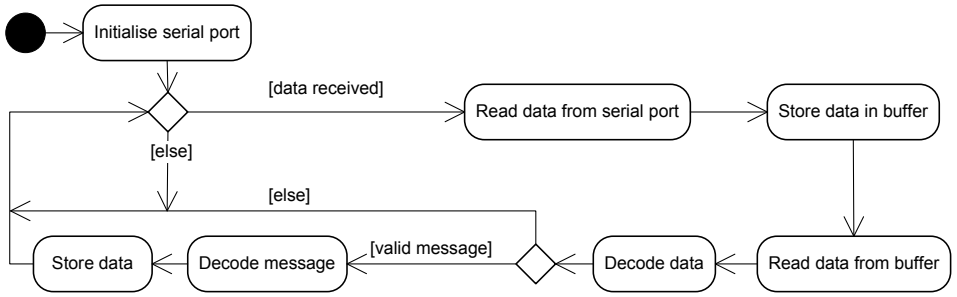


Figure 4.7: Comm port activity diagram

Table 4.1 gives a brief overview of the various identifiers in the makeup of the message packet.

| **Identifier** | **Description** |
|---|---|
| $ | Start of string delimiter |
| II | Two matching letters uniquely identifying the message (in this case 'I') |
| Data | Binary data string of arbitrary length |
| * | End of string delimiter |
| CS | 1 byte checksum |

Table 4.1: Protocol identifiers

The construction of a packet must comply with the following rules.

- The checksum is performed between the start of string and end of string delimiters.
- The checksum should be replaced with a '+' character if the result of the checksum is equal to the start of string or end of string delimiter.
- If a data byte is equal to a start of string or end of string delimiter, it should be queued twice to avoid confusion.

### Telemetry Packet

The data is broken into bytes. Each byte is sent individually. The least significant byte of a data value is sent first. Table 4.2 shows the order in which the data is sent. Once received the bytes need to be recombined to form the correct data values.

| Data | Format | Size (Bytes) | Gain | Unit |
|---|---|---|---|---|
| Latitude | Signed 32 bit Int | 4 | $10^{-7}$ | Degrees |
| Longitude | Signed 32 bit Int | 4 | $10^{-7}$ | Degrees |
| Altitude | Signed 32 bit Int | 4 | $10^{-7}$ | Degrees |
| Roll | Signed 16 bit Int | 2 | $\frac{2\pi}{2^{15}}$ | Radians |
| Pitch | Signed 16 bit Int | 2 | $\frac{2\pi}{2^{15}}$ | Radians |
| Yaw | Signed 16 bit Int | 2 | $\frac{2\pi}{2^{15}}$ | Radians |

Table 4.2: Data packet from avionics equipment

#### Gain

The data needs to be multiplied by a gain shown in Table 4.2 to reveal its correct value in its specified unit.

### 4.5.3 Aircraft Satellite Emulator Software

#### Calculation Engine (FU3)

The calculation engine functional block (FU3) performs various calculations. However, the calculations are all related to the flight path geometry. This is the flight path past a ground station, as specified by the flight strategy presented in the previous section. Figure 4.8 illustrates the geometry, which is applicable to both a satellite and an aircraft, the only difference being the altitude. Point $B$ is the point when the airborne object starts to communicate with the ground station. This usually occur when the airborne object is at the horizon relative to the ground station. $S_3$ represents the distance the object travels as it moves from point $B$ to $E$, when it is at the maximum elevation angle. Note that $S_3$ is half the flight route distance the object travels during a single pass.

The calculations are done according to the following principle: First calculate $S_1$ then $S_2$ and $S_3$. Because the velocity of the airborne object is known and $S_3$ is calculated, the time of the flight path past the ground station can be calculated.

$S_1$ is calculated as follows:

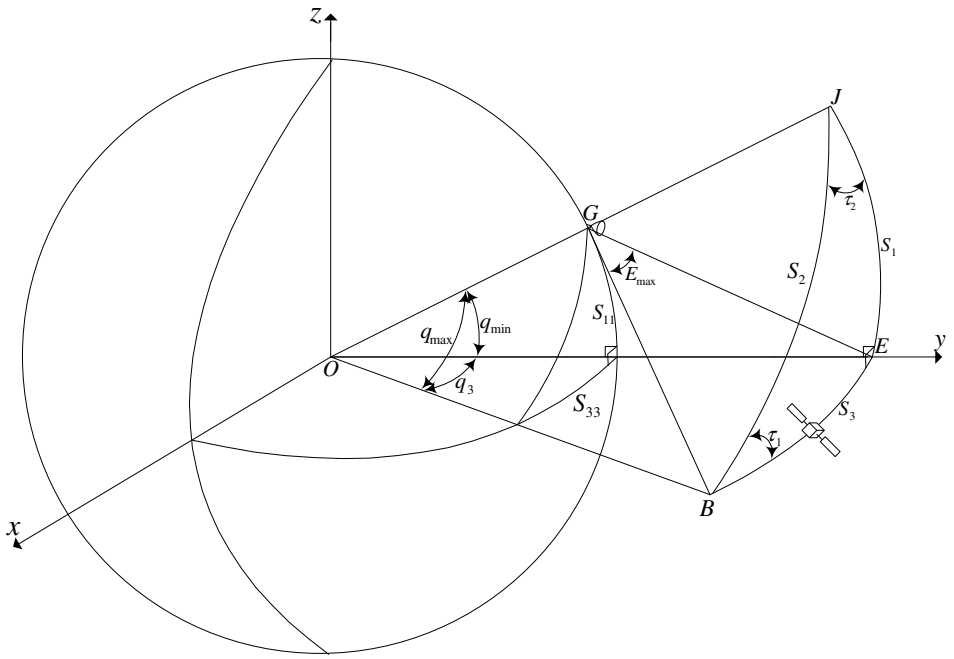


Figure 4.8: Flight path geometry

- First calculate the minimum direct line of sight distance by using the cosine rule in triangle $OGE$. See section 2.6.5 for the equation.
- Then calculate the minimum geocentric angle $q_{min}$. This is the geocentric angle when the airborne object is closest to the ground station, which is calculated by applying the cosine rule in triangle $GOE$. See section 2.6.6 for the equation.
- $S_1$ can then be calculated by the following equation

$$S_1 = q_{min} \cdot (R + h) \tag{4.5.1}$$

where $R$ is the sum of the earth's radius and ground station altitude and $h$ is the airborne object's altitude.

$S_2$ is calculated as follows:

- First calculate the maximum direct line of sight distance by using the cosine rule in triangle $OGB$. See section 2.6.5 for the equation.
- Then calculate the maximum geocentric angle $q_{max}$. This is the geocentric angle at which the airborne object first starts to communicate with

the ground station, which The angle is calculated by applying the cosine rule in triangle $GOB$. See section 2.6.6 for the equation.

- $S_2$ can then be calculated by the following equation

$$S_2 = q_{max} \cdot (R + h) \tag{4.5.2}$$

where $R$ is the sum of the earth's radius and ground station altitude and $h$ is the altitude of the airborne object.

$S_3$ is calculated as follows:

- First calculate the angle $\tau_1$ by using the sine rule in spherical triangle $EBJ$.

$$\tau_1 = \arcsin\left(\frac{\sin S_1}{\sin S_2}\right) \tag{4.5.3}$$

- The following two equations are derived by making use of the cosine law in spherical triangle $EBJ$

$$\cos S_3 = \cos S_2 \cdot \cos S_1 + \sin S_2 \cdot \sin S_1 \cdot \cos \tau_2 \tag{4.5.4}$$

$$\cos \tau_2 = -\cos \tau_1 \cdot \cos 90^\circ + \sin \tau_1 \cdot \sin 90^\circ \cdot \cos S_3 \tag{4.5.5}$$

In both equations there are two unknown parameters $\tau_2$ and $S_3$

- $S_3$ is calculated by substituting the two equations into each other.

$$S_3 = \arccos\left(\frac{\cos S_2 \cdot \cos S_1}{1 - \sin S_2 \cdot \sin S_1 \cdot \sin \tau_1}\right) \tag{4.5.6}$$

Thus, by calculating the total distance the airborne object travels, the total time of the flight pass can be calculated by dividing the total distance by the velocity of the airborne object.

$S_{11}$ is the distance from the ground station to the nadir when the satellite is closest to the ground station. $S_{33}$ is the distance on the ground that the satellite travels. These two distances are calculated as follows:

$$S_{11} = q_{min} \cdot R \tag{4.5.7}$$

$$S_{33} = q_3 \cdot R \tag{4.5.8}$$

Figure 4.9 shows the calculated area activity diagram of the calculation engine (FU3). The calculations presented in this section are used in this module to calculate the area under the elevation-time graph. The time the airborne object takes to complete a pass is first calculated. The time is then divided into much smaller intervals. At each interval the new position of the airborne object is calculated and, thereafter, the elevation angle. After all the elevation angles have been calculated, the area must be calculated. This is done by looping through the intervals and calculating the area of each elevation-time interval. The total area is then calculated as the sum of all the area intervals.

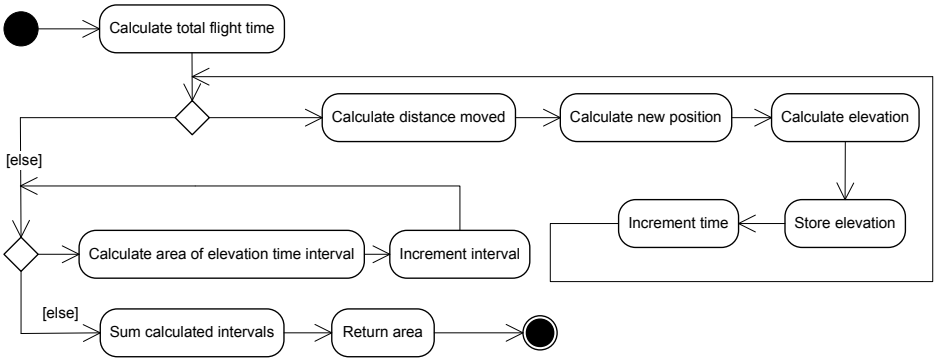


Figure 4.9: Calculate area activity diagram

The altitude at which the aircraft must fly to emulate the orbital characteristics of a satellite as closely as possible is calculated by an iterative method. Figure 4.10 shows the activity diagram of this process. The altitude is calculated by comparing the elevation-time graph of the aircraft at various altitudes to that of the satellite elevation-time graph. The delta area is then calculated by subtracting the areas of one graph from the other. The smaller the result after the subtraction, the better the two graphs match and the aircraft thus emulates the satellite orbital characteristics closer more closely.

## 4.6 SAA Control

The purpose of the control software on the OBC is to control the steerable antenna array in order to direct the steerable antenna beam in the direction of a specific ground station. The antenna beam can be directed by making use

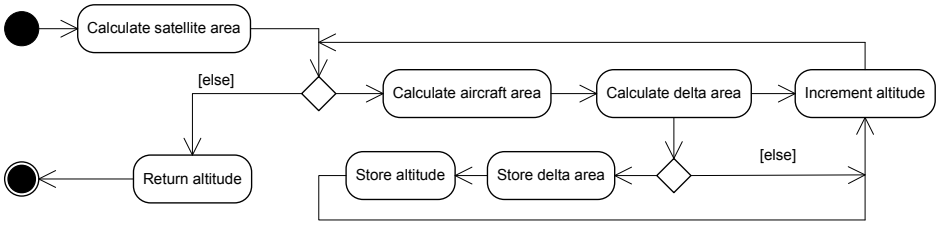


Figure 4.10: Calculate aircraft altitude activity diagram

of the aircraft and ground station coordinates and the attitude of the aircraft. Some of the SAA control module functional units will be discussed in more detail in the following subsections.

### 4.6.1 Control Unit (FU13)

The control unit controls the program and data flow of the SAA control software. Figure 4.11 presents the activity diagram of the control unit (FU13). The scheduling interface, CAN, serial port and SAA are initialised at the start of execution. The coordinates of the ground station to which the SAA needs to be directed are read from the scheduling software. The satellite data required by the control software will be requested from the ASE. The data received should be multiplied by its specific gain to give the correct value in the unit. The SAA control algorithm will make use of this data to calculate the $\phi$ (phi) and $\theta$ (theta) angles needed to direct the antenna beam to a specific ground station. The control signals will then be transmitted to the steerable antenna array.

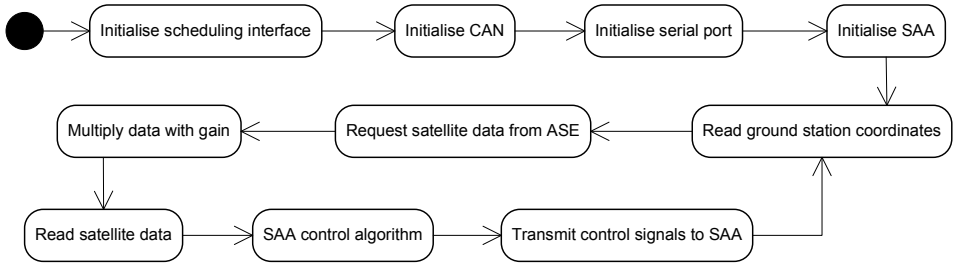


Figure 4.11: Control unit (FU13) activity diagram

### 4.6.2 Virtual CAN Node (FU12)

The CAN driver is responsible for the sending and receiving of data on the CAN bus. The CAN driver implements a virtual CAN node that makes use of the existing CAN driver on the SH4. The virtual CAN node can be seen as an interface between an application on the OBC and the actual CAN node connected to the CAN bus. The protocol is implemented by the CAN controller of the actual CAN node. See section 4.7 for a description of the CAN protocol. Figure 4.12 shows a graphical representation of the layers between an application on the SH4 and the CAN bus. A description of the different layers will now be given:

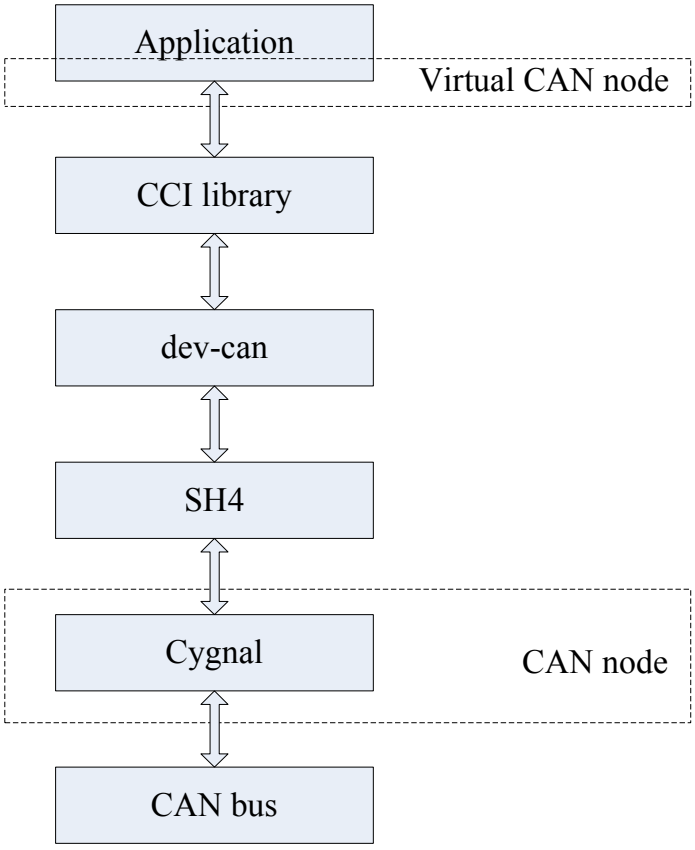


Figure 4.12: Model of CAN interface

**CAN bus** The CAN bus is responsible for the physical transmission of bits.

**Cygnal** Is an 8-bit processor in the CAN node. It is connected to the CAN bus. It handles all low level telemetry and telecommands. All CAN packets are handled by this processor.

**SH4** On-board computer

**dev-can** Is the CAN driver on the QNX operating system and it handles all the CAN packets received or transmitted by the SH4.

**CCI Protocol** CCI Protocol is a C library for QNX applications that contains all the necessary tools to create a CAN interface (virtual node) for an user application. It contains functions that simplify the implementation of telemetry and telecommands. It also handles the link between the application and the CAN device driver (dev-can).

**Application** Runs on the SH4. The virtual node implemented in this application enables it to connect to the CAN bus.

### 4.6.3 Scheduling Interface (FU14)

The scheduling interface is a thread created by the control unit (FU13) that continiously requests the target ground station coordinates from the scheduling software (FU15). Note, however, that the program flow will block until a reply is received from the scheduling software. Figure 4.13 shows the activity diagram of this thread.

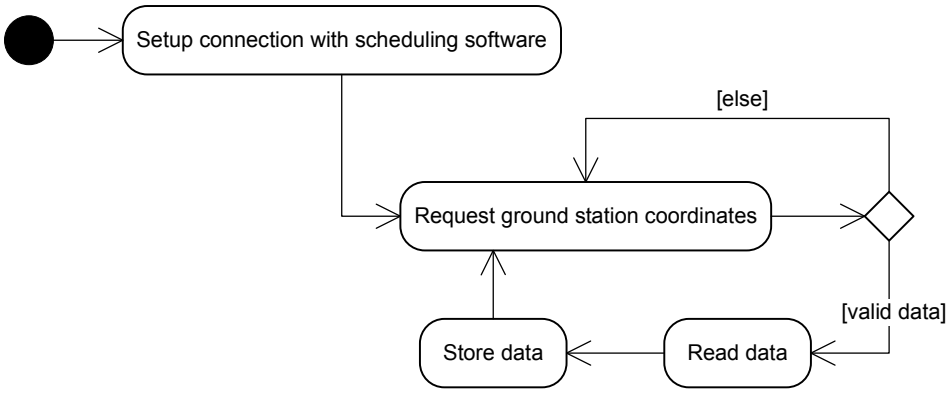


Figure 4.13: Scheduling Interface (FU14) activity diagram

### 4.6.4 Calculate Steering Angles (FU16)

The task of this functional unit is to calculate the necessary steering angles to direct the antenna beam towards the specified target ground station. The functional unit uses the current position and attitude of the airborne object to direct the antenna beam.

Two steering angles theta, $\phi$ and phi $\theta$, are used to direct the antenna beam and are defined as follows: The $\theta$ angle is measured from the positive x-axis of the body frame towards the projection of the target vector on the x-y plane. The $\theta$ angle is measured from the positive z-axis of the body frame, towards the target vector.

Figure 4.14 shows the define angles with the steerable antenna mounted on the bottom of the plane. Although the figure illustrates an aircraft as the airborne object, this functional unit and subsequent calculations is also applicable to the implementation on a satellite.

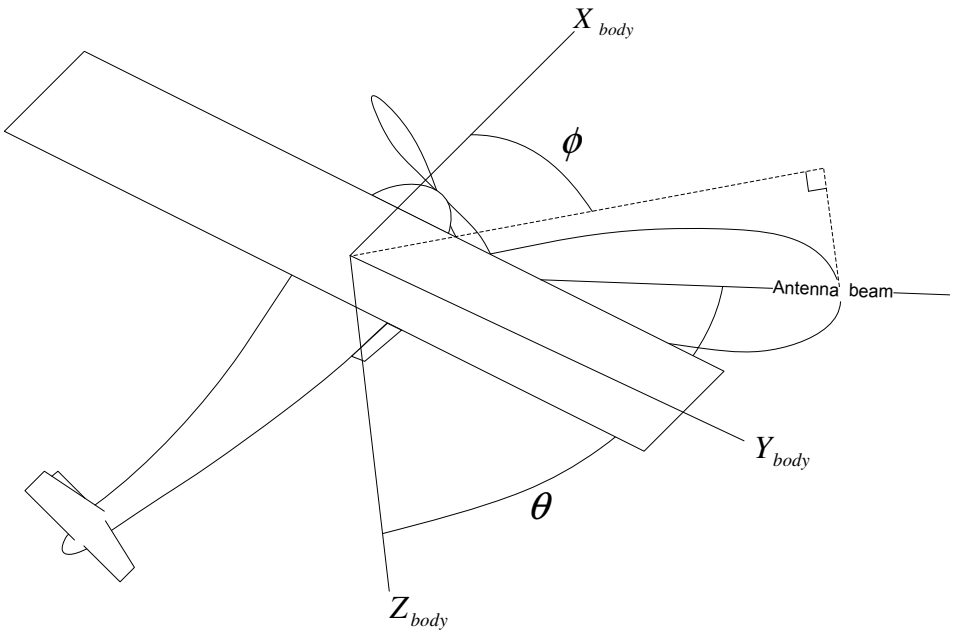


Figure 4.14: Steering angle definitions

### 4.6.5 SAA Interface (FU17)

The interface between the control software and the steerable antenna array (SAA) is a serial interface, which emits control commands that control the $\phi$ (phi) and $\theta$ (theta) angles of the antenna. By means of this, control is possible to direct the antenna beam (Figure 4.14) in a specified direction.

## 4.7 CAN Protocol

The CAN protocol must conform to that of the local satellite constructor SunSpace. Documents [23] and [24] specify the CAN protocol used by SunSpace. From these documents it was found that SunSpace uses the ISO898 standards that describe the CAN physical layer. These standards are based on the Bosch CAN2.0 specifications [25]. The ISO11898-1 [26] standard specifies the CAN data link layer and physical signalling. ISO11898-2 [27] specifies the implementation of high-speed data transmission. The CAN network, as specified by SunSpace, operates at 1 Mbit/s. This section provides a brief overview of the basic CAN protocol, by specifying the CAN message format and the custom CAN messages used to communicate between the ASE and the payload.

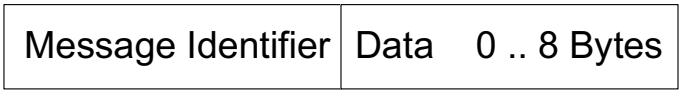


Figure 4.15: CAN message

A CAN message (Figure 4.15) consists of two main parts namely, a message identifier and a data field. The message identifier is made up of 29 bits while the data field may vary between 0 and 8 bytes. The message identifier is then subdivided into three parts called the frame identifier, the source address and the destination address (Table 4.3). The frame identifier (FID) specifies the message type and the priority of the message. A lower FID number has a higher priority. The message type consists of two parts, the main type and the sub-type. The main type is made up of the 5 most significant bits of the frame identifier and is used to determine the main message type. The remaining 8

bits, called the sub-type, define a specific command of the main message type.

| **Frame Identifier (MSB)** | **Source Address** | **Destination Address (LSB)** |
|---|---|---|
| 13 bits | 8 bits | 8 bits |

Table 4.3: Message Identifier format

A unique 8 bit address or ID is assigned to each CAN node. The address is used in the source and destination address fields of the frame identifier. The destination and source address fields indicate to and from which node the CAN message is sent.

By using the message type telemetry it is possible for the payload to access the required data from the aircraft satellite emulator (ASE) via the CAN bus. A telemetry message can only be sent to a specific node on the CAN bus. The data transfer is initiated by a telemetry message containing a request. The CAN node that receives the request replies with a telemetry message containing the data. Table 4.4 shows the frame identifier format (FID) of a telemetry message. The requested data, called a telemetry frame, is returned in the data field of the CAN message. Data items, also referred to as telemetry channels, are grouped to form telemetry frames. Thus a telemetry frame may contain more than one item. The telemetry CAN message specifies in the field sub-type of the telemetry frame identifier (Table 4.4) which telemetry frame is being requested. The data field of a CAN message can contain 8 bytes, thus the telemetry channels must be grouped into telemetry frames of 8 bytes or less. Table 4.5 specifies the grouping of the telemetry channels into telemetry frames.

| **Message type** | **Sub-type** |
|---|---|
| 5 bits | 8 bits |
| 0x0A - Telemetry request<br>0x0B - Telemetry reply | 0x00 - 0xFF |

Table 4.4: Telemetry frame identifier format

| Telemetry frame number | Telemetry frame description | Size (Bytes) |
|---|---|---|
| 0x00 | Latitude | 4 |
| | Longitude | 4 |
| 0x01 | Altitude | 4 |
| 0x02 | Roll | 2 |
| | Pitch | 2 |
| | Yaw | 2 |

Table 4.5: Grouping of telemetry frames

The PCAN-USB module creates an USB interface that enables a PC and a CAN bus to connect. By using the routines and the device drivers supplied by the PCAN-USB module it is possible to link the aircraft satellite emulator (ASE) to the CAN bus. The PCAN-USB module creates a CAN node.

| Message Identifier | Data bytes | | | | | | | | Comments |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| 0x0A000A05 | | | | | | | | | SH4 request frame 0 from ASE |
| 0x0B00050A | Latitude | | | | Longitude | | | | ASE reply with frame 0 to SH4 |
| 0x0A010A05 | | | | | | | | | SH4 request frame 1 from ASE |
| 0x0B01050A | Altitude | | | | | | | | ASE reply with frame 1 to SH4 |
| 0x0A020A05 | | | | | | | | | SH4 request frame 2 from ASE |
| 0x0B02050A | Roll | | Pitch | | Yaw | | | | ASE reply with frame 2 to SH4 |

Table 4.6: Telemetry messages

CAN node IDs of 0x0A and 0x05 were assigned to the virtual CAN node on the SH4 of the payload and the ASE CAN node, respectively. Table 4.6 shows the format of the telemetry messages implemented that will allow data to be transmitted from the ASE to the payload.

| Message type | Sub-type |
|---|---|
| 5 bits | 8 bits |
| 0x06 - Telecommand request<br>0x07 - Telecommand acknowledge | 0x00 - 0xFF |

Table 4.7: Telecommand frame identifier format

| Message Identifier | Data bytes | | | | | | | | Comments |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| 0x0601050A | | | | | | | | | ASE send telecommand to SH4 |
| 0x07010A05 | | | | | | | | | SH4 send acknowledge to ASE |

Table 4.8: Telecommand messages

The SAA control software application on the payload can be disabled by a telecommand. Table 4.7 shows the frame identifier format of a telecommand. The ASE is thus able to stop the execution of the SAA control software on the SH4 by sending a telecommand. Table 4.8 specifies this specific telecommand and the reply message that is sent if the telecommand is received.

## 4.8 Transmission Link

Section 3.2 describes the transmission link strategy adopted. The main concepts of this strategy are that:

- Free space losses should be compensated for to emulate the transmission link.
- The free space loss compensation is achieved by attenuation of the received or transmitted signal.
- The attenuation will be applied at the ground station.
- For the reason mentioned in section 3.2, Doppler compensation is not considered. Doppler compensation will, however, be applied at the ground station when the system is implemented on an actual satellite.
- Flight tests will be performed in wide-open areas to minimise the effect of terrain scattering.

### 4.8.1 Transmission Calculations

See Appendix A for the results of the link budget calculations. This link budget calculated a satellite link. For the emulation on board an aircraft the transmission link should be attenuated to achieve the same free space losses

as would be the case with the satellite link. The satellite link budget can therefore be seen as the reference link budget.

The following subsections will discuss the various parameters of the link budget in more detail.

### Attenuation

The transmission link can be emulated by using an attenuator to adjust the power of the received or transmitted signal. Ideally the up-link signal would be attenuated at the aircraft but, in order not to complicate the KU Leuven design, the decision was made to attenuate the signal at the ground station. Therefore, to implement this strategy the attenuation for a specific flight route must be calculated beforehand. This information will enable the ground station to attenuate the transmission link. It is crucial that the ground station attenuate the transmission link by the right amount at the right time and that the ground station is synchronised with the aircraft at the start of its flight route.

Table 4.9 shows the losses caused by the distance between the ground station and the aircraft at various maximum elevation angles. Losses between the ground station and the satellite is displayed in the last column of Table 4.9. Thus the amount of attenuation can be calculated by subtracting the satellite free space losses from those of the aircraft at a specific altitude and maximum elevation angle.

| Free space loss (dB) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Altitude (km)** | 0.05 | 0.345 | 0.64 | 0.935 | 1.23 | 1.525 | 1.82 | 2.115 | 2.41 | 2.705 | 3 | 500 |
| **0 deg elevation** | 128.09 | 136.48 | 139.16 | 140.81 | 142 | 142.93 | 143.7 | 144.35 | 144.92 | 145.42 | 145.87 | 168.26 |
| **10 deg elevation** | 89.231 | 106 | 111.36 | 114.65 | 117.02 | 118.88 | 120.41 | 121.71 | 122.84 | 123.84 | 124.73 | 164.63 |
| **20 deg elevation** | 83.344 | 100.12 | 105.49 | 108.78 | 111.16 | 113.02 | 114.56 | 115.86 | 116.99 | 117.99 | 118.89 | 161.58 |
| **30 deg elevation** | 80.046 | 96.822 | 102.19 | 105.48 | 107.86 | 109.73 | 111.26 | 112.57 | 113.7 | 114.7 | 115.6 | 159.22 |
| **40 deg elevation** | 77.864 | 94.641 | 100.01 | 103.3 | 105.68 | 107.55 | 109.08 | 110.39 | 111.52 | 112.53 | 113.42 | 157.45 |
| **50 deg elevation** | 76.34 | 93.117 | 98.484 | 101.78 | 104.16 | 106.03 | 107.56 | 108.87 | 110 | 111 | 111.9 | 156.13 |
| **60 deg elevation** | 75.275 | 92.052 | 97.419 | 100.71 | 103.09 | 104.96 | 106.5 | 107.8 | 108.94 | 109.94 | 110.84 | 155.17 |
| **70 deg elevation** | 74.566 | 91.343 | 96.71 | 100 | 102.38 | 104.25 | 105.79 | 107.09 | 108.23 | 109.23 | 110.13 | 154.52 |
| **80 deg elevation** | 74.158 | 90.935 | 96.303 | 99.595 | 101.98 | 103.84 | 105.38 | 106.69 | 107.82 | 108.82 | 109.72 | 154.15 |
| **90 deg elevation** | 74.025 | 90.802 | 96.17 | 99.462 | 101.84 | 103.71 | 105.25 | 106.55 | 107.69 | 108.69 | 109.59 | 154.03 |

Table 4.9: Free space loss calculations

### Noise Floor

The following parameter values for the link are known:

$$T_{ant} = 290\,\text{K} \quad (4.8.1)$$
$$B = 50\,\text{kHz} \quad (4.8.2)$$
$$F = 1.5\,\text{dB} \quad (4.8.3)$$

The receiver noise temperature can be calculated by Equation 2.10.4

$$T_{rec} = (10^{0.15} - 1)290 \quad (4.8.4)$$
$$= 119.636\,\text{K} \quad (4.8.5)$$

and the $T_{tot}$ by using Equation 2.10.3

$$T_{tot} = 290 + 119.636 \quad (4.8.6)$$
$$= 409.636\,\text{K} \quad (4.8.7)$$

Thus the received noise power is calculated by Equation 2.10.2 to be

$$P_{NoiseFloor} = (1.38 \times 10^{-23})(409.636)(50 \times 10^{3}) \quad (4.8.8)$$
$$= -125.488\,\text{dBm} \quad (4.8.9)$$

This noise floor value is, however, optimistic, it does not take into account other losses in the receiver. It was found that $-110\,\text{dBm}$ is a more realistic rating for the noise floor at the carrier frequency of $2.4\,\text{GHz}$, given the specifications of the typical ISM band hardware.

### Signal Strength

The received power is calculated by considering all the gains and losses of the transmission link. The gains are added to the transmit power and the losses subtracted. Equation 4.8.10 expresses this in terms of decibels (dB).

$$P_{RX}\ (dB) = P_{TX} + G_{TX} - L_{FTX} - L_{FS} + G_{RX} - L_{FRX} - L_{Pol} \quad (4.8.10)$$

### Transmit Power

Constraints such as the available power, especially for the satellite, have to be considered before specifying the transmit power of the transmitter. It is also important that the transmit power is high enough to allow successful communication.

It was found that a transmitter with a power rating of 10 W is sufficient to allow reliable communication for the up and downlink.

### Signal to Noise Ratio

The SNR is the ratio, at the receiver, of the received power to the received noise. This ratio gives a indication of communication link reliability. The SNR is calculated by Equation 4.8.11.

$$SNR = \frac{P_{RX}}{P_{NoiseFloor}} \tag{4.8.11}$$

## 4.9 Conclusion

The architecture and system-wide design decisions made, were discussed in this chapter. The system architecture comprised of two main components, the emulator and satellite payload module. The components were then further divided into functional units, the architecture of which were presented and discussed. The implementation of the system design as presented in this chapter will be discussed next.

# Chapter 5

# Implementation

The implementation of system design of the previous chapter will now be presented. This again constitutes the implementation of two main components, the ASE and SAA control module respectively. A more detailed description of the implementation of components contained in these two modules will now be given. Refer to Figure 4.1 on page 41 for an overview of the system.

## 5.1 Aircraft Satellite Emulator

### 5.1.1 Project Overview

Figure 5.1 shows all the ASE project files. These files were used to create the ASE software application. A brief description of each file will follow.

**Air_Interface** The Builder environment generates this file automatically when a project is created.

**Backgnd** This code generates a background frame which occupies the whole screen area. New frames are superimposed on this frame, which acts solely as a background. The code also creates and starts the clock thread.

**Main** The main ASE project file.

**Thread_Clock** This thread is used solely for timing purposes. It makes use of the multi-media timer routines and is accurate in terms of time measurement. This routine measures time in 100 ms chunks. Timer variables which may be used by routines for time measurement are incremented.

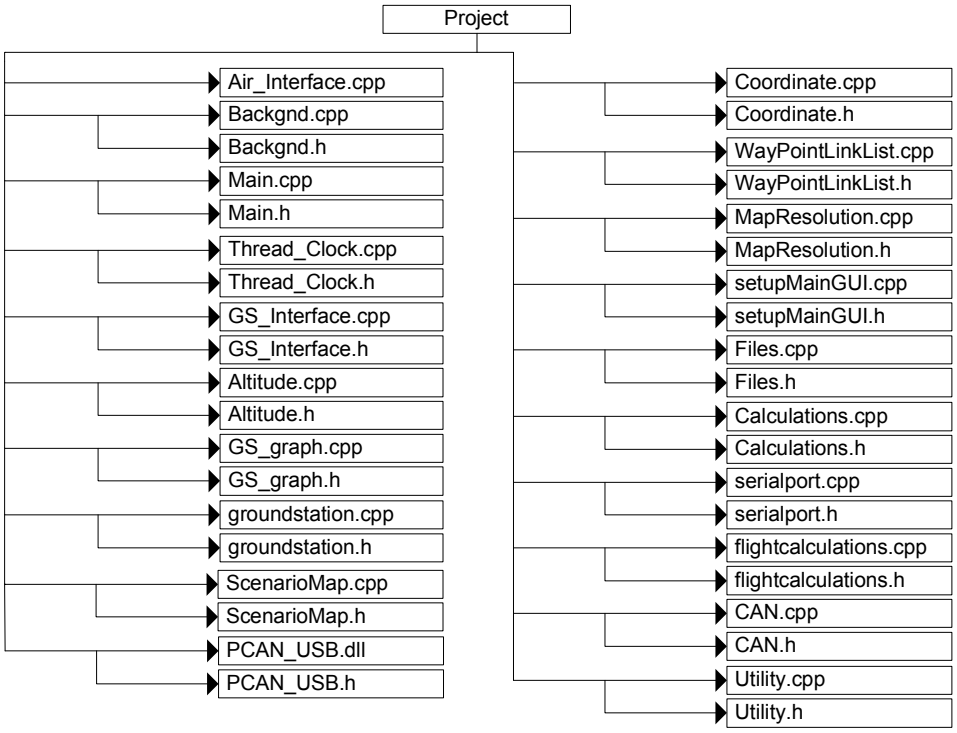


Figure 5.1: Project files

**GS_Interface** The ground station window is invoked with this code each time a ground station is selected by means of a button on the main application window. The text boxes are grouped into two sections, ground station and aircraft parameters. Text boxes enable the user to specify parameters for each. After the maximum elevation and velocity has been specified, the remaining aircraft parameters (altitude above ground station and distance from ground station) will be calculated by this application.

**Altitude** This code will provide an altitude window. This will enable the user to specify the altitude of the flight route at specific points of the flight route.

**GS_graph** This file enables the display of graphs of a selected ground station. Elevation and azimuth time graphs of the calculated satellite and aircraft flights and also the actual aircraft flight are displayed.

**groundstation** The groundstation class stores various parameters associated with a ground station.

**ScenarioMap** The ScenarioMap class is responsible for the scenario map. This entails the scenario map resolution, drawing and displaying the scenario setup by the user.

**PCAN_USB.dll and Pcan_usb.h** PCAN_USB.dll is a dynamic link library provided by PEAk-System. The dll provides functions that enable an application to control the PCAN-USB device. The Pcan_usb.h file contains descriptions of the functions provided.

**Coordinate** The Coordinate class is used by various classes throughout this project. The class store a coordinate point by storing the latitude, longitude and altitude coordinate.

**WayPointLinkList** The WayPointLinkList class makes use of the c++ standard template library (stl) containers to implement a link list. This list is used in this project to store the flight route coordinates.[28]

**MapResolution** This code sets the scenario map resolution.

**setupMainGUI** This class sets the placement of GUI items such as buttons and panels on the main GUI window.

**Files** The Files class is used to write data to files.

**Calculations** The Calculations class performs various calculations that are related to the flight path geometry. The class is used to calculate the appropriate aircraft parameters to allow the aircraft to emulate a satellite pass.

**serialport** The functions that control the serial port are all house in the serialport class.

**flightcalculations** The in-flight calculations are performed by this class.

**CAN** The functions form part of the CAN interface are housed in this class.

**Utility** The Utility class houses data structures used by various classes throughout this project.

The finished ASE software will be loaded on the emulator PC by loading the ASE software folder. This folder contains within it three separate folders

with the software executable and dll. Figure 5.2 shows this file hierarchy, with the files contained in each folder. The three folders are the FlightRoute, GroundStations and Telemetry folder. The files contained in each folder will be discussed next.

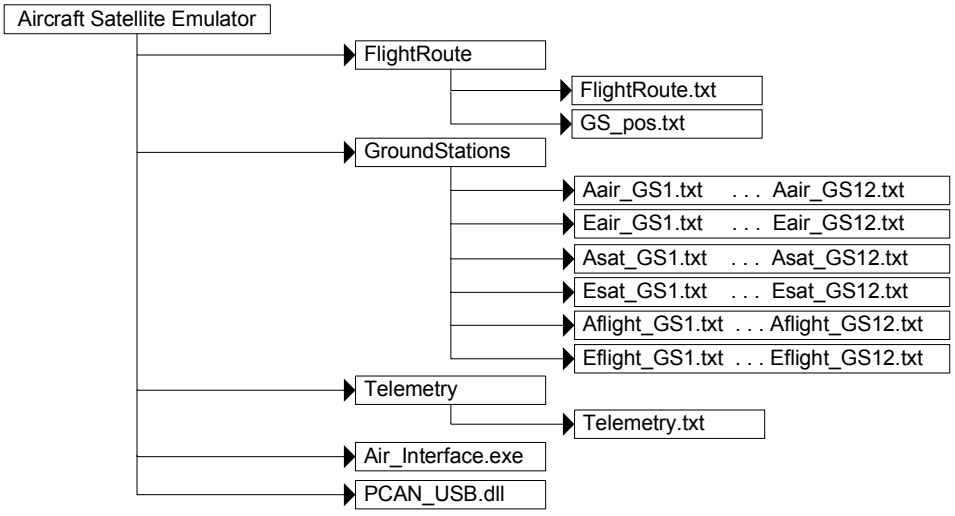


Figure 5.2: ASE software file hierarchy

### FlightRoute folder

**FlightRoute.txt** The indicated flight route is stored in this file.

**GS_pos.txt** The selected ground station coordinates are stored in this file.

### GroundStation folder

**Air_GS1.txt ... Air_GS12.txt** Stores the predicted azimuth angles for a specific aircraft flight path past various ground stations.

**E_GS1.txt ... E_GS12.txt** Stores the predicted elevation angles for a specific aircraft flight path past various ground stations.

**Asat_GS1.txt ... Asat_GS12.txt** Stores the predicted azimuth angles for a specific satellite flight path past various ground stations.

**Esat_GS1.txt ... Esat_GS12.txt** Stores the predicted elevation angles for a specific satellite flight path past various ground stations.

**Aflight_GS1.txt ... Aflight_GS12.txt** Stores the calculated azimuth angles for a specific aircraft flight path past various ground stations.

**Eflight_GS1.txt ... Eflight_GS12.txt** Stores the calculated azimuth angles for a specific aircraft flight path past various ground stations.

**Telemetry folder**

**Telemetry.txt** Stores the aircraft avionics data received.

### 5.1.2 Project Functional Units

The grouping of classes into functional units are presented in this section. Figure 5.3 illustrates this grouping with the functions contained in each class. Note that *get* and *set* functions are not shown.

### 5.1.3 Project Class Overview

A butterfly diagram Figure 5.4 provides a brief overview of the classes.

### 5.1.4 ASE-Payload Interface

This section discusses the various interfaces between the ASE and payload. The functional unit and class applicable to the relevant interface is indicated next to each subsection.

#### Button_init_PCANClick (FU9 CAN)

The PCAN-USB device is initialised by the user pressing a button. The Button_init_PCANClick function in the CAN class is then called to initialise the device. The following procedure is followed in this function to initialise the PCAN-USB device. Figure 5.5 shows a flow diagram of this process. The libraries necessary to operate the PCAN-USB device are contained in the dynamic link library (dll) file provided by PEAk-System. The PCAN_USD.dll is loaded by acquiring a handle to the dll. If the handle was successfully opened acquires the function handles to all the functions provided by the library. The

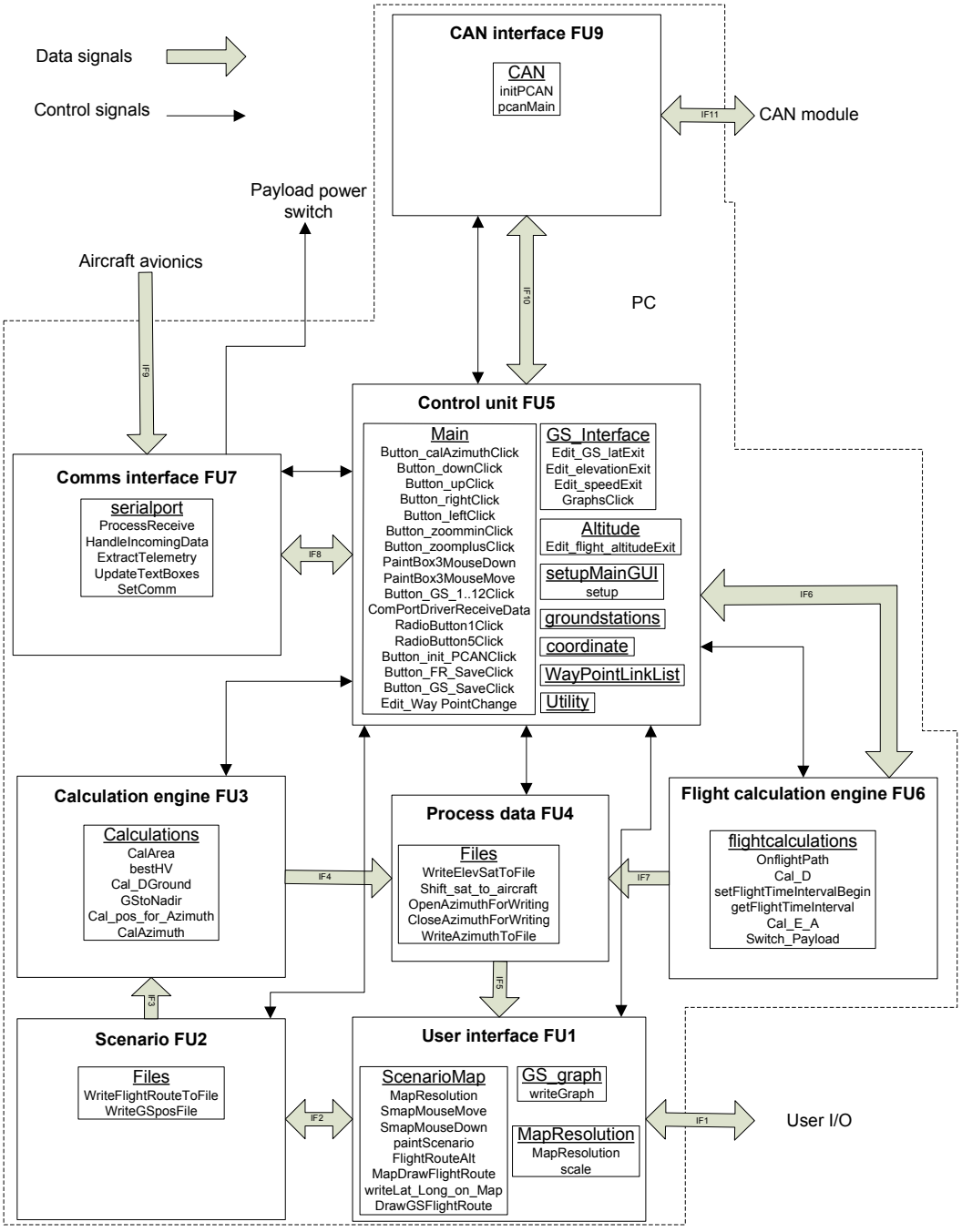


Figure 5.3: Grouping of classes into functional units

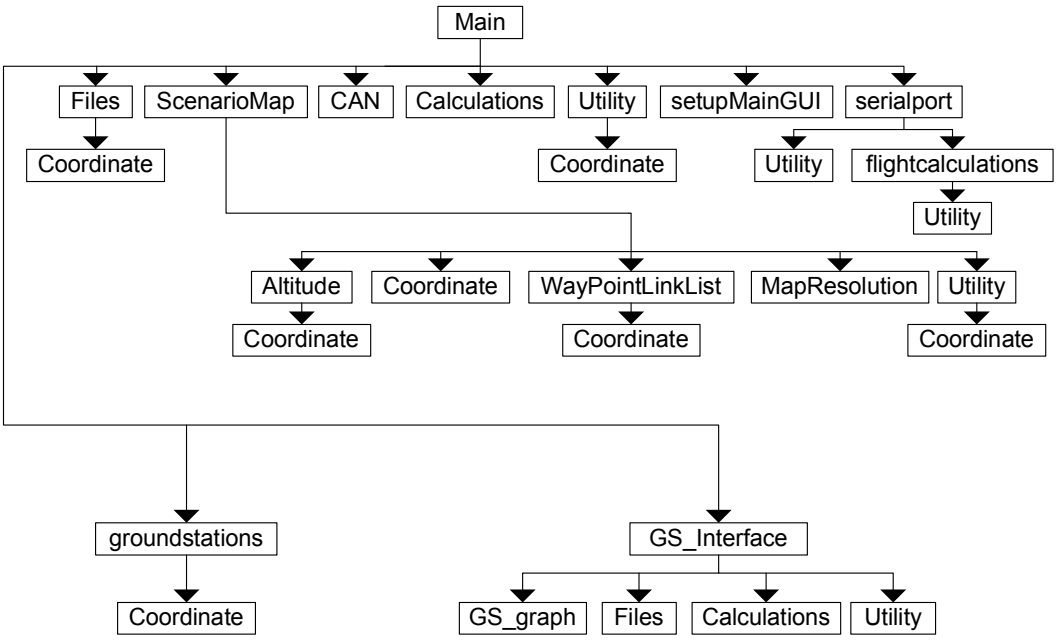


Figure 5.4: Butterfly diagram of project classes

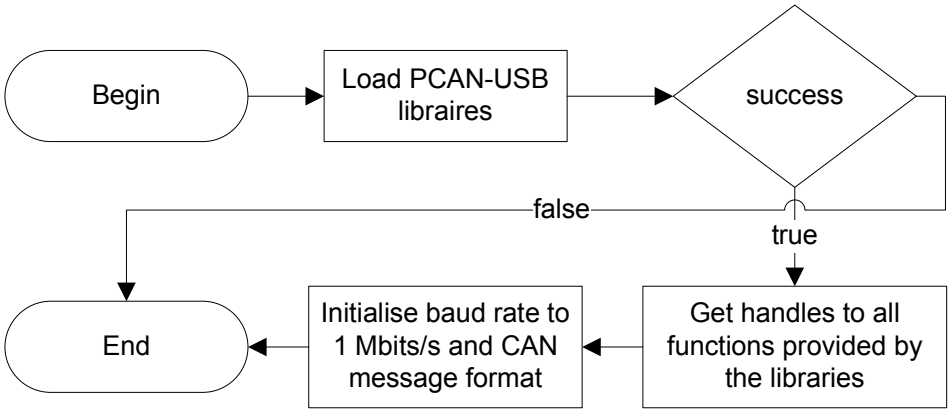


Figure 5.5: Initialise CAN function flow diagram

baud rate of the PCAN-USB device is then set to 1 Mbit/s and the CAN message format specified. The CAN message format is set to the extended frame format (29-bit message identifier).

### pcanMain (FU9 CAN)

This function implements the CAN interface of the ASE with the payload. The CAN interface (FU9) comprises the driver function. This driver function

is implemented as a thread (Timer1Timer) that executes every 50 milliseconds and calls the pcanMain function. Figure 5.6 shows a flow diagram of the implemented function.

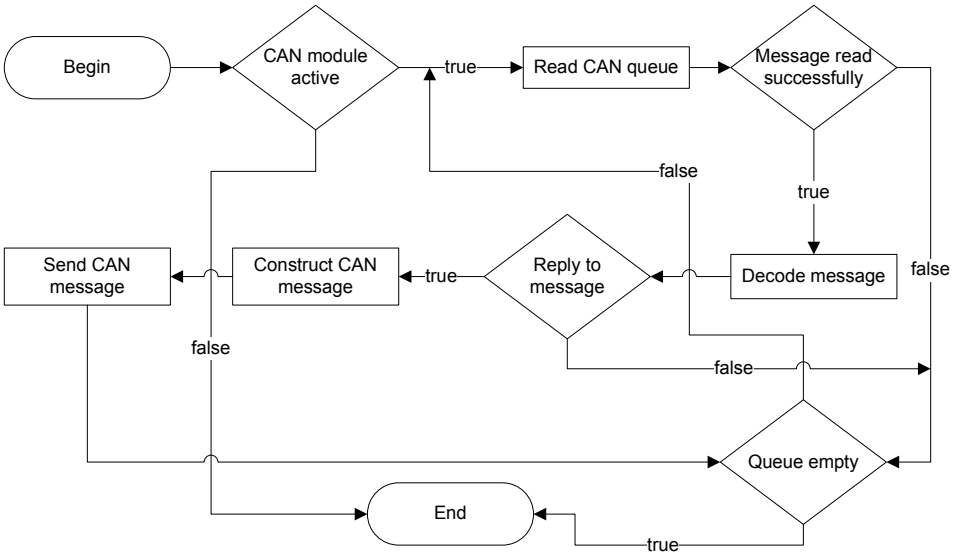


Figure 5.6: CAN driver flow diagram

Every 50 milliseconds this function starts by first trying to read a CAN message from the queue of the CAN module. CAN messages received by the CAN module are stored in a message queue. If the queue is read successfully, the function decodes the message by assessing the CAN message identifier. If it is found that the CAN message requests data from the ASE, the function constructs the reply message with the requested data. A reply is then sent in the form of the constructed CAN message. This process will be repeated until the CAN message queue is empty. The function then ends. Note that specific data is requested by the CAN message, specifying a particular telemetry frame. See Table 4.5 for the grouping of the telemetry frames and Table 4.6 for the telemetry message format.

### SetComm (FU7 serialport)

The serial interface with the payload is initialised by the SetComm function. The SetComm function is part of the Comms interface (FU7). The task of this serial interface is to switch the payload on or off. The SetComm function

is called each time the user initialises the serial port to the ASE by selecting a serial port by means of radio buttons provided by the GUI.

#### Switch payload on or off (FU7)

The serial port to the payload switches the payload on or off by switching a relay that controls the power to the payload. The payload is switched on or off by setting the voltage on the DTR and RTS pins of the serial port high or low respectively.

The voltage on the DTR and RTS pins are set by the EscapeCommFunction function, which is provided by C++ Builder. The function enables the control of specific pins on the serial port. The EscapeCommFunction function is called by the flight calculation engine (FU6) via the control unit (FU5) to switch the payload on or off. The flight calculation engine calculates when the payload must be switched on or off.

### 5.1.5 Aircraft Avionics Interface

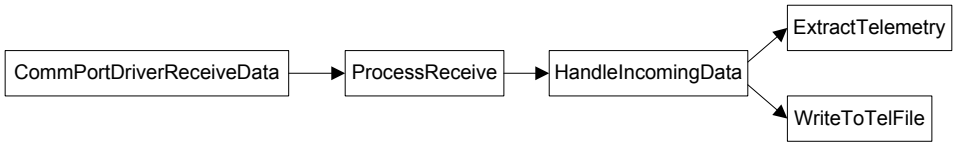


Figure 5.7: Overview of the function used to implement the aircraft avionics equipment driver

The aircraft avionics equipment driver is responsible for reading the telemetry data that is sent serially by the aircraft avionics equipment. The driver forms part of the Comms interface functional unit (FU7) and is housed in the serialport class. Various telemetry packets are transmitted serially by the aircraft avionics equipment via the COM port. It is thus the task of the driver to distinguish the different telemetry packets and store the telemetry data desired by the application.

The driver was implemented with the help of a C++ Builder open course component called TComport. This component controls the serial port and stores data received at the COM port in a buffer. A function (in this case

CommPortDriverReceiveData) is then called by the component, with a pointer to the data in the buffer and the amount of data in the buffer as parameters.

It is then necessary to read the data in the buffer, process the data and store the telemetry data if a desired packet has been received. The functions used to implement this process are shown as a diagram in Figure 5.7.

### Initialise Serial Port (FU7)

The serial port to the aircraft avionics equipment needs to be initialised by the user. Com port 1 to 4 is opened by selecting the corresponding radio button. By selecting a radio button is a function called that operates in the following way: The function first checks the com port and disconnects it if it is already connected. The selected com port is then connected. A check is performed and a message is displayed to indicate success or failure.

### CommPortDriverReceiveData (FU5 Main)

The task of this function is to extract the data contained in the buffer of the TComport component and to call another function to process the data. The CommPortDriverReceiveData function is called by the TComport component. A pointer to the data in the buffer of the TComport component and the amount of data in the buffer are received by this function as parameters.

The CommPortDriverReceiveData function is implemented by using a *for* loop and the pointer to the data in the buffer to read the data from the buffer one byte at a time. The ProcessReceive function is called each time a byte is read. The task of the ProcessReceive function is to process the data read. This process will continue till all the bytes have been read from the buffer. Figure 5.8 shows the flow diagram of the implementation of this function.

### ProcessReceive (FU7 serialport)

The function of this process is to determine whether if a valid telemetry packet has been received. This is done by checking that the received data corresponds to the implemented protocol.

The ProcessReceive function is invoked by the CommPortDriverReceive-Data function each time a byte is read from the COM port buffer. The byte

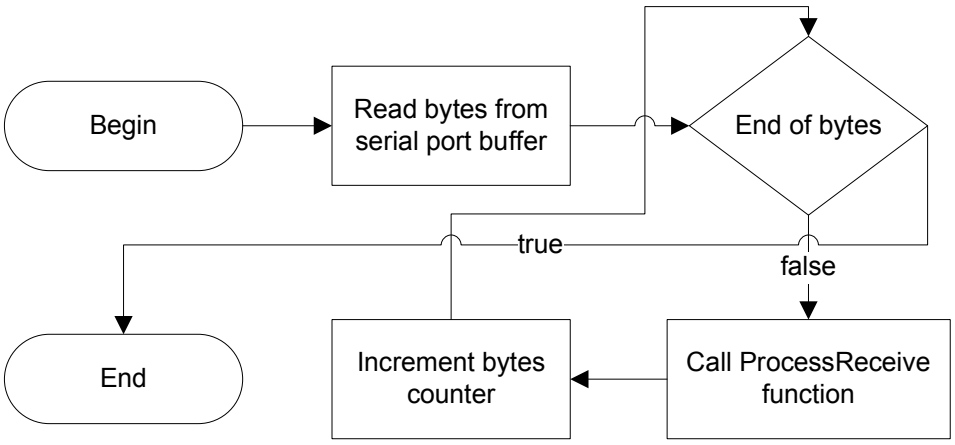


Figure 5.8: CommPortDriverReceiveData function flow diagram

that has been read is received as an input parameter to the ProcessReceive function.

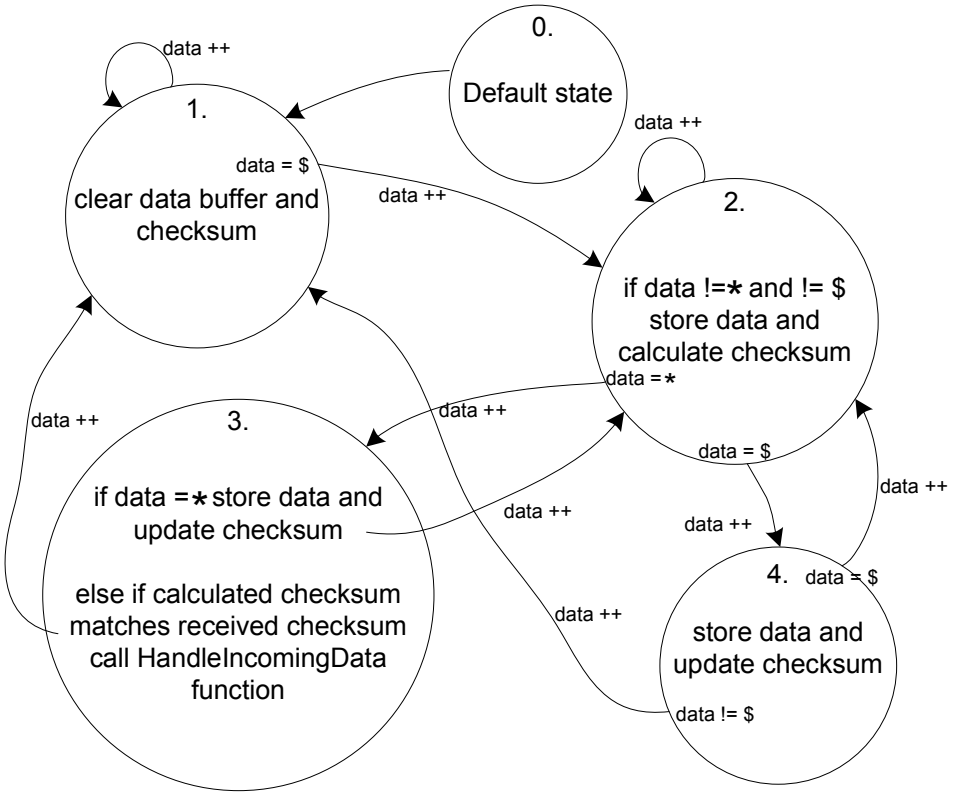


Figure 5.9: State diagram of ProcessReceive function

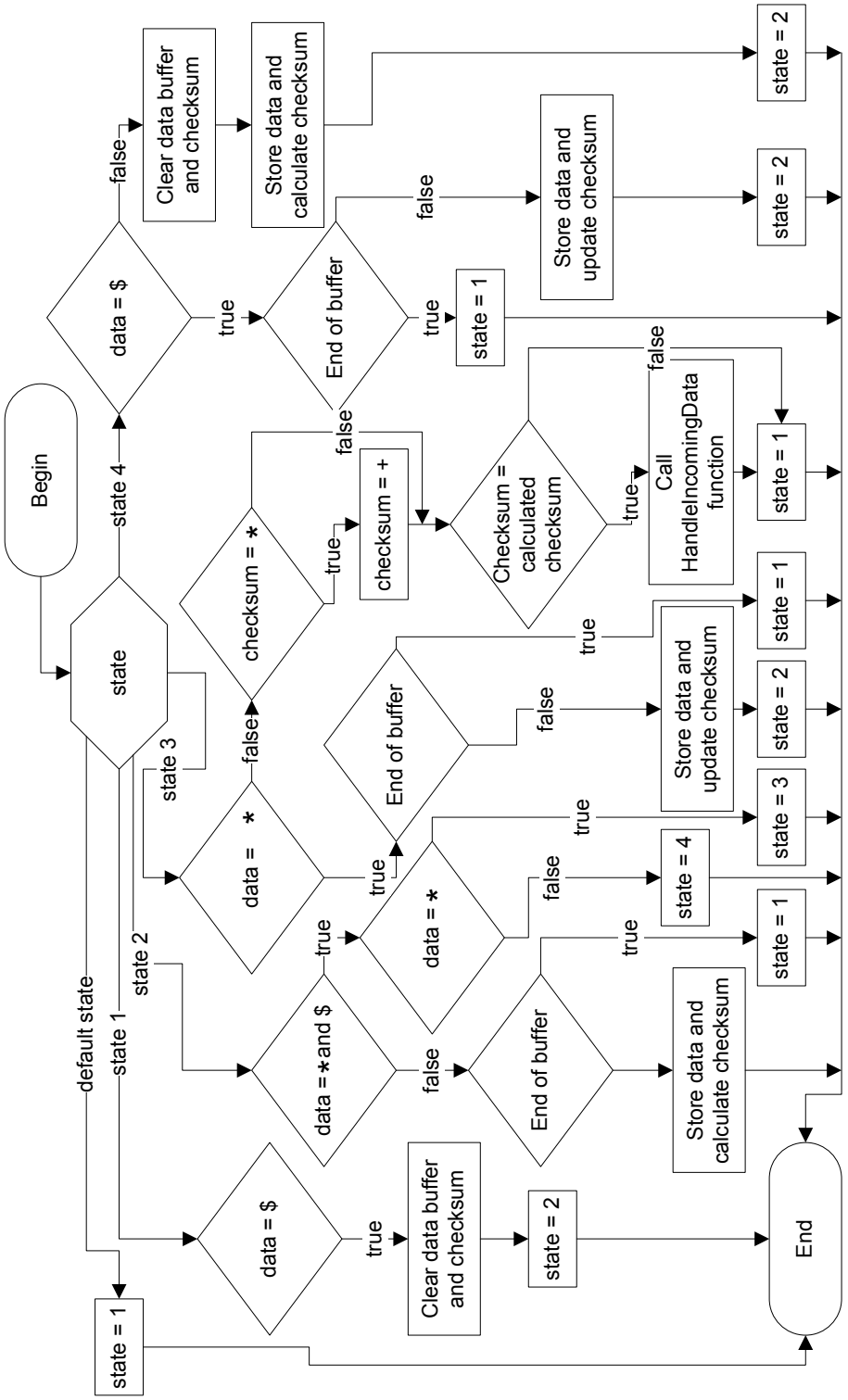


Figure 5.10: ProcessReceive function flow diagram

The implementation of this function can be thought of as a state machine. See Figure 5.9 for a graphical representation of this state machine and Figure 5.10 for a flow diagram of the implementation. Four different states are used and they are implemented by a switch statement. A default state will execute the first time this function is called. The task of this state is to direct the program to the first state.

The task of the first state is to check for the *start of string* delimiter ($). If the delimiter has been received this state will direct the program to the second state. Otherwise it will remain in this state until it receives a *start of string* delimiter. The *start of string* delimiter '$' indicates the beginning of a telemetry packet.

The second state will store the received data in a buffer and calculate the checksum. This will continue till a *start* '$' or *end* '∗' *of string* delimiter is received. This second state will then direct the program to states four or three respectfully.

According to the protocol data must be queued twice if it is equal to a *start* '$' or *end* '∗' *of string* delimiter. It is for this reason that state three will first check whether an *end of string* delimiter '∗' has been received. If this is true, this state stores the *end of string* delimiter '∗' as data and updates the calculated checksum. State three then directs the program back to state two. If no *end of string* delimiter '∗' was received, the task of this state is to determine if the checksum is correct. This is done by matching the calculated checksum with the received one. A valid telemetry packet has been received and the HandleIncomingData function is called if the two checksums concur. If this is not the case, an error has occurred and the received data is discarded. After both cases the program is diverted back to state one.

State four is invoked by state two if a *start of string* delimiter '$' is received in state two. As previously mentioned, data must be queued twice if it is equal to the *start of string* delimiter '$'. The task of state four is to determine whether data equal to a *start of string* delimiter '$' has been queued twice or whether the *start of string* delimiter '$' received in state two was the beginning of a new telemetry packet. This is done by checking whether a *start of string* delimiter '$' was received in state four. If this is true, the *start of string* delimiter '$' is stored as data in a buffer and the checksum updated. A new telemetry packet has arrived if no *start of string* delimiter '$' is received in

state four. The buffer is then cleared by storing the new data in the first position of the data buffer. State four will direct the program back to state two after both these cases.

### HandleIncomingData (FU7 serialport)

The task of the HandleIncomingData function is to identify the telemetry packets required by the ASE application. The HandleIncomingData function is called each time a telemetry packet is received and successfully parsed. A special telemetry packet that contains all the necessary information required by the ASE was constructed for this project. Thus the task of the HandleIncomingData function is to identify this packet and call the appropriate functions to process the received data.

Two characters uniquely identify a specific message packet. The first two characters parsed and saved in a buffer by the previous ProcessReceive function are these message identifiers. Two matching 'I' characters were used in this project to identify the message packet used by the ASE.

### ExtractTelemetry (FU7 serialport)

This function extracts the stored data bytes to form the data values sent by the aircraft avionics equipment. A data structure was created to store these values. The data structure created enables access to each byte and data value individually. See Figure 5.11 for the declaration of the data structure. The data is read into this data structure by looping through the stored bytes and writing the byte read into the correct position in the data structure.

### WriteToTelFile (FU7 serialport)

The WriteToTelFile function stores the data received from the aircraft in a file. By storing the aircraft data it is possible to review the flight off line. Note, however, that the data values must first be multiplied by a gain of a certain value to give the correct value in its specified unit. See Table 5.2 for these gain values.

```
typedef struct
{  int              latitude;              // byte  0...3: Word 0
   int              longitude;             //  "    4...7:  "   1
   int              altitude;              //  "    8..11:  "   2
   short            roll;                  //  "   12..13:  "   3
   short            pitch;                 //  "   14..15:  "   4
   short            yaw;                   //  "   16..17:  "   5
 }packet_struct;

typedef struct
{ union
  {
      packet_struct data;
      char bite[sizeof(packet_struct)];
    }sel;
}data_struct_type_t;
```

Figure 5.11: Data structure

### 5.1.6 Aircraft Satellite Emulator Software

The various functions that form part of the ASE software will be discussed next. The functional unit and class of which each function is part is indicated next to the function. Refer to page 55 for a more detailed description of the parameters and calculations performed in the calculation engine (FU3).

#### MapResolution (FU1 ScenarioMap)

Invokes the MapResolution function of the MapResolution class that sets the scenario map resolution.

#### SMapMouseMove (FU1 ScenarioMap)

This function is invoked when the user moves the mouse. The task of this function is to draw the flight path for a selected ground station according to the position of the mouse relative to the selected ground station.

#### SMapMouseDown (FU1 ScenarioMap)

The SMapMouseDown function handles the actions when the mouse is pressed.

### paintScenario (FU1 ScenarioMap)

This function draws the scenario setup by the user on the scenario map. This includes the ground station position, flight path past ground stations, selected flight path and aircraft position.

### FlightRouteAlt (FU1 ScenarioMap)

This function determines if the user has clicked on a way-point and if this is the case invokes the function the altitude form to enable the user to specify the altitude at that specific way-point.

### MapDrawFlightRoute (FU1 ScenarioMap)

The MapDrawFlightRoute function draws the flight route specified by the user and store the selected way-points in a list.

### WriteLat_Long_on_Map (FU1 ScenarioMap)

The task of the WriteLat_Long_on_Map function is to write latitude and longitude coordinates on the side of the scenario map.

### DrawGSFlightRoute (FU1 ScenarioMap)

The DrawGSFlightRoute function draws the flight route way-points on the scenario map. These way-points can be seen as spheres located at the start and end of the flight route past a ground station. The way-points indicate to a user that the aircraft has reached the start or end of a flight path. The way-points are initially red on the scenario map. However a way-point will turn green if the aircraft is located within one of these way-points. This indicates that the aircraft has reached the start of the flight path. The second way-point of that flight path will also turn green when the aircraft reaches it. This function then turns both way-points back to red after the aircraft has moved out of the second way-point (and thus out of the flight path).

This implementation can be thought of as a state machine. In the initial state the state machine turns both way-points red. When the aircraft reaches a way-point the state machine proceeds to the second state that turns the way-point that has been reached to green. The state machine will proceed to

the final state when the aircraft reaches the second way point. In this state the state machine turns both way-points to green. The state machine will revert back to the initial state that turns both way-point to red, when the aircraft moves out of the second way-point.

### writeGraph (FU1 GS_graph)

This function generates elevation and azimuth graphs by reading the calculated data from a file into a series. The series is then displayed to the user when the user invokes the ground station graphs.

### MapResolution (FU1 MapResolution)

The MapResolution function sets the scenario map resolution.

### scale (FU1 MapResolution)

The scale function changes the viewing prospective of the scenario map. The function allows the user to move the scenario map up, down, left, right, zoom in and zoom out.

### WriteFlightRouteToFile (FU2 Files)

The WriteFlightRouteToFile function writes the flight route coordinates to a file. This function is called by the Button_FR_SaveClick function that is invoked if the user selects the *Save to File* button.

### WriteGSposFile (FU2 Files)

This function writes the selected ground station coordinates to a file.

### CalArea (FU3 calculations)

This function calculates the area beneath the elevation-time graph. Figure 5.13 shows a flow diagram of this function. The aircraft velocity and altitude and the ground station altitude and maximum elevation angle are given as input parameters to this function. The function first calculates the total flight time for the flight past a ground station as specified by the received parameters. Figure 5.12 is a flow diagram of this part of the function. The function first

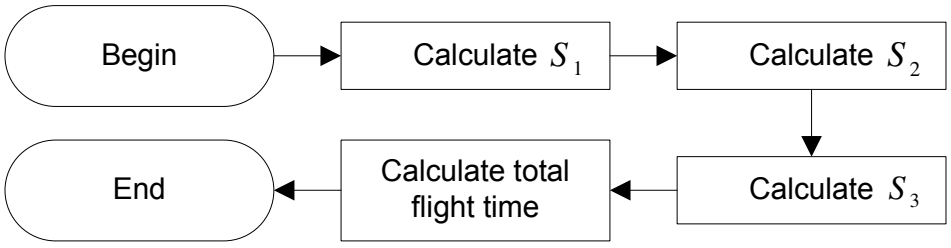


Figure 5.12: Flow chart of CalTotalFlightTime function

calculates the $S_1$ then the $S_2$ and finally the $S_3$ parameter. The $S_3$ parameter is half the distance the aircraft travels in the specified flight path past a ground station. The total time of a flight pass is then calculated by multiplying the distance $S_3$ by 2 and dividing the result by the velocity of the aircraft. The function then divides the total flight time into increments. At each increment the function calculates and stores the aircraft elevation angle. The function then proceeds to a second loop where the elevation-time area is calculated. After this the function returns the calculated area.

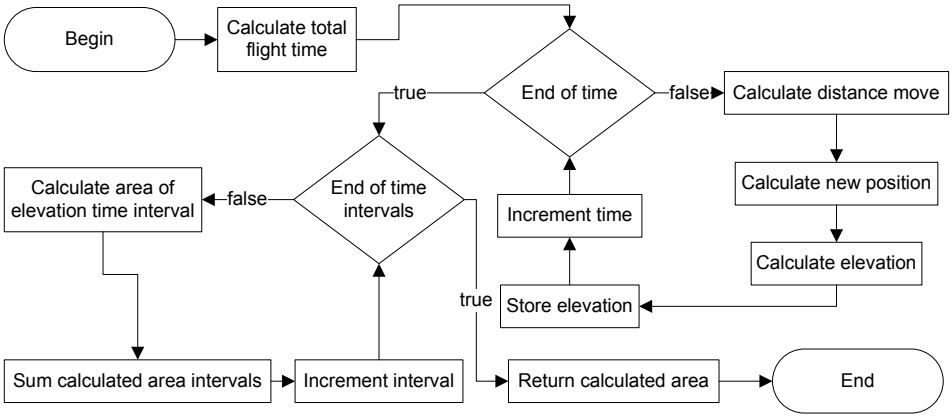


Figure 5.13: Flow chart of CalArea function

### bestHV (FU3 Calculations)

The task of the bestHV function is to return the altitude that would enable the aircraft to emulate the satellite characteristics as closely as possible. The input parameters to this function are the aircraft velocity, aircraft maximum elevation angle, calculated satellite area and ground station altitude. Figure 5.14

shows a flow diagram of this function. An iterative method is implemented to calculate the appropriate altitude. The altitude is calculated by incrementing the altitude in intervals and then comparing various elevation-time windows to the satellite elevation-time window. The function calculates the area of each elevation -time window by calling the CalArea function. The result is then subtracted from the satellite elevation-time area. The smaller the result the better the two graphs match and the better the aircraft conforms to the satellite characteristics.

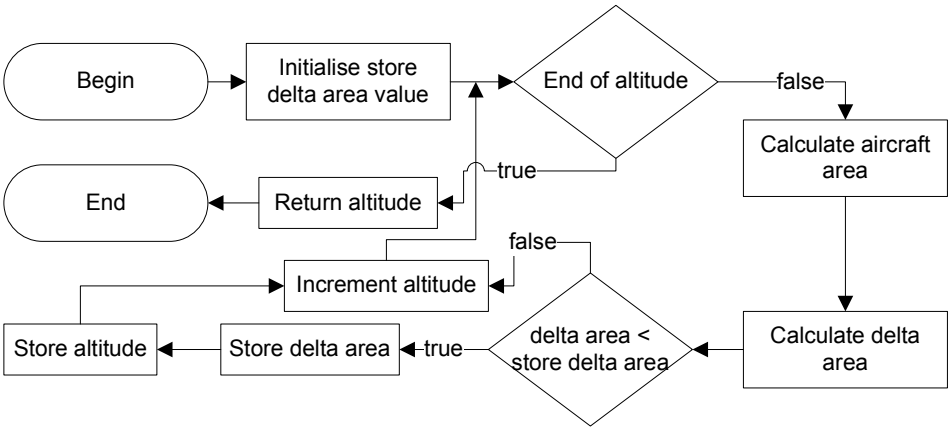


Figure 5.14: Flow diagram of bestHV function

### Cal_DGround (FU3 Calculations)

This function calculates the distance of the flight path on the ground. FU1 uses the distance to map the flight path. The task of this function is thus to calculate $S_{22}$ and then to multiply the result by 2 to get the total distance. See Figure 4.8 on page 56 and also the equations derived from this figure for a description of the equations used in this function.

The inputs to this function are the aircraft velocity, aircraft altitude, aircraft maximum elevation angle and ground station altitude. The ground distance is calculated by first calculating $S_1$, $S_2$ and $S_3$, in that specific order. Thereafter it is possible to calculate $S_{22}$. The result is then multiplied by 2. The function returns the result after the multiplication.

### GStoNadir (FU3 Calculations)

The GStoNadir function calculates the distance from the ground station to the nadir point at which the aircraft is closest to the ground station. This distance is denoted by $S_{11}$ in Figure 4.8 on page 56. An overview of the calculations used to calculate $S_{11}$ are presented in Section 4.5.3 where the calculation engine FU3 is described.

The aircraft altitude, aircraft maximum elevation angle and ground station altitude are the input parameters to this function. The function then calculates the distance from the ground station to the nadir point $S_{11}$ and returns the result.

### Cal_pos_for_Azimuth (FU3 Calculations)

This function calculates the start and end coordinates of the flight path. A brief overview of the calculations used will be presented in this section.

Figure 5.15 shows the flight geometry used to calculate the start and end coordinates of the flight path. Point $G$ is the ground station position. The line $GT$ is the distance in terrestrial degrees from the ground station to the nadir point at which the aircraft is closest to the ground station. This distance is calculated by the GStoNadir function in FU3. The line $RS$ is the flight path of the aircraft on the ground. The task of the function Cal_pos_for_Azimuth is to calculate the coordinates of the points $R$ and $S$. Points $R$ and $S$ being the start and end positions of the flight path. These points are calculated by using the $TA$, $GA$, $RM$ and $MT$ distances and the orientation of the flight path specified by the angle $g$, which describes the orientation of the flight path to the ground station. The angle $g$ is specified by the GUI. The user specifies this angle by clicking on the ground station generated on the map and moving the flight path to the desired orientation. The calculations used to calculate $TA$, $GA$, $RM$ and $MT$ are presented next.

For the spherical triangle $GTA$ the following results hold:

$TA$ is calculated by using the law of sines

$$\frac{\sin g}{\sin TA} = \frac{\sin 90}{\sin GT} \tag{5.1.1}$$

$$TA = \arcsin(\sin GT \sin g) \tag{5.1.2}$$

where $g$ is the angle specified by the GUI and $GT$ is calculated by the GStoNadir function.

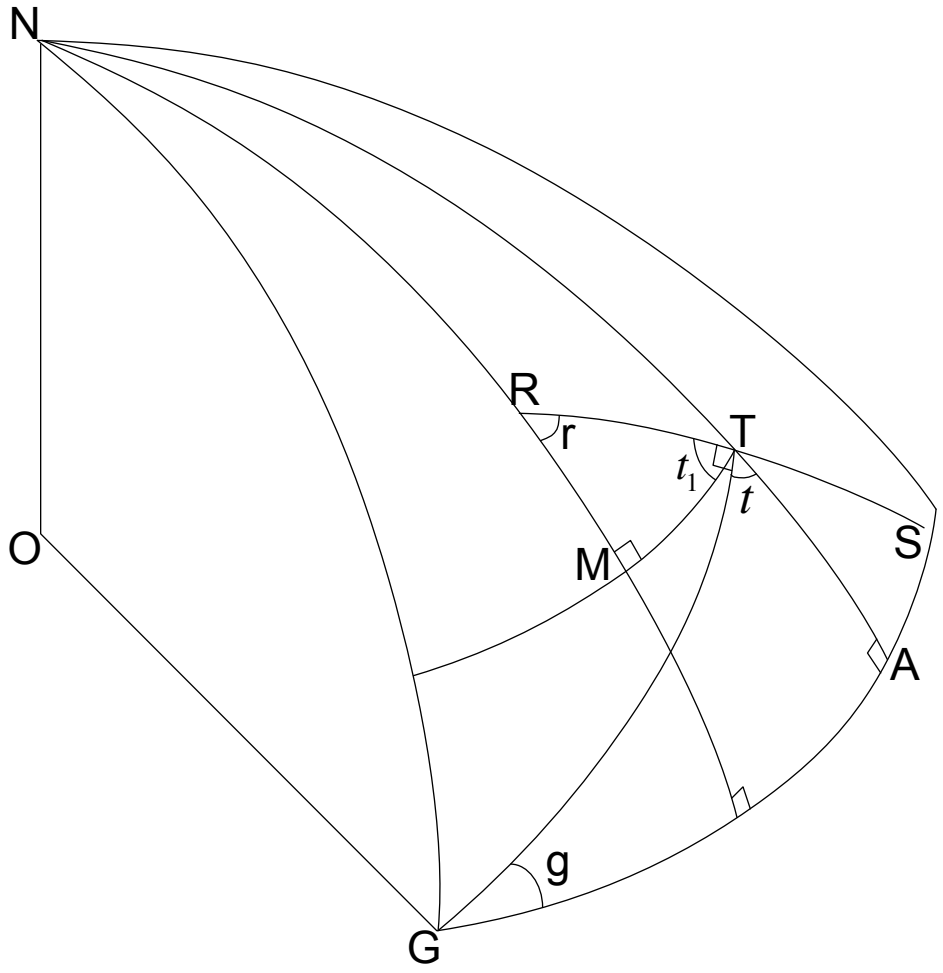


Figure 5.15: Flight path geometry

$GA$ is calculated by using the law of cosines

$$\cos GA = \cos GT \cos TA + \sin GT \sin TA \cos t \tag{5.1.3}$$

and

$$\cos t = -\cos g \cos 90^\circ + \sin g \sin 90^\circ \cos GA \tag{5.1.4}$$

$$\cos t = \sin g \cos GA \tag{5.1.5}$$

Equation 5.1.4 is then substituted into Equation 5.1.3 to give the following result

$$\cos GA = \cos GT \cos TA + \sin GT \sin TA \sin g \cos GA \tag{5.1.6}$$

$$GA = \arccos\left(\frac{\cos GT \cos TA}{1 - \sin GT \sin TA \sin g}\right) \tag{5.1.7}$$

For the spherical triangle $RTM$ the following results hold:

$RM$ is calculated by using the law of sines

$$\frac{\sin t_1}{\sin RM} = \frac{\sin 90}{\sin RT} \tag{5.1.8}$$

$$\frac{\sin(90^\circ - g)}{\sin RM} = \frac{\sin 90}{\sin RT} \tag{5.1.9}$$

$$RM = \arcsin(\sin(90^\circ - g)\sin RT) \tag{5.1.10}$$

where $RT$ is the distance of the flight path on the ground calculated by the Cal_DGround function.

$MT$ is calculated by using the law of cosines

$$\cos MT = \cos RT \cos RM + \sin RT \sin RM \cos r \tag{5.1.11}$$

and

$$\cos r = -\cos t_1 \cos 90^\circ + \sin t_1 \sin 90^\circ \cos MT \tag{5.1.12}$$

$$\cos r = \sin t_1 \cos MT \tag{5.1.13}$$

Equation 5.1.12 is then substituted into Equation 5.1.11 to give the following result

$$\cos MT = \cos RT \cos RM + \sin RT \sin RM \sin t_1 \cos MT \tag{5.1.14}$$

$$MT = \arccos\left(\frac{\cos RT \cos RM}{1 - \sin RT \sin RM \sin(90^\circ - g)}\right) \tag{5.1.15}$$

Figure 5.16 shows the flow diagram that uses the calculations presented in this section to implement the Cal_pos_for_Azimuth function. Note that the calculation of the coordinates of points R and S differs according to the quadrant of the flight path relative to the ground station.

### CalAzimuth (FU3 Calculations)

The azimuth angles are calculated by this function.

### WriteElevSatToFile (FU4 Files)

The task of this function is to write the elevation angles calculated by FU3 to a file.

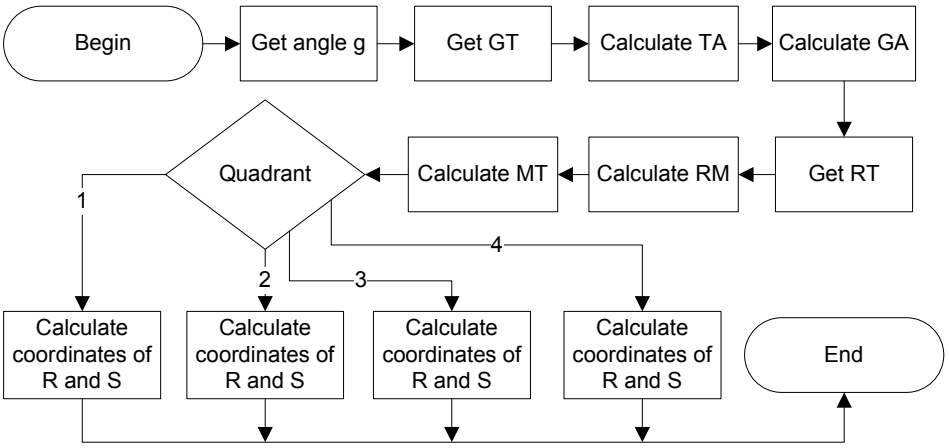


Figure 5.16: Flow chart of Cal_pos_for_Azimuth function

### shift_sat_to_aircraft (FU4 Files)

As explained in Section 3.3, the elevation graphs are shifted to align the maximum elevation angles. The shift_sat_to_aircraft function shifts the satellite data for this maximum elevation angle alignment of graphs.

The input parameters to this function are the file names of the aircraft and satellite data calculated by FU3. The function refers to chart 1 as the aircraft data and chart 2 as the satellite data. As mentioned, the task of the function is to shift the data of chart 2 onto chart 1, enabling the maximum elevation angles to align.

Figure 5.17 shows the flow diagram of the implementation of this function. The function starts by opening the data file of chart 1. Then the function reads through the data file and finds the maximum elevation angle and the corresponding time value. The function will then repeat this for the data of chart 2. The time shift is then calculated by subtracting the two saved time values at maximum elevation. The data of chart 2 is then shifted by changing the time data according to the time shift.

### Edit_GS_latExit (FU5 GS_Interface)

The Edit_GS_latExit function is invoked each time the user exits the Lat, Long and Altitude text boxes from the ground station window. See Figure 5.21. The function then proceeds to check that a valid ground station coordinate

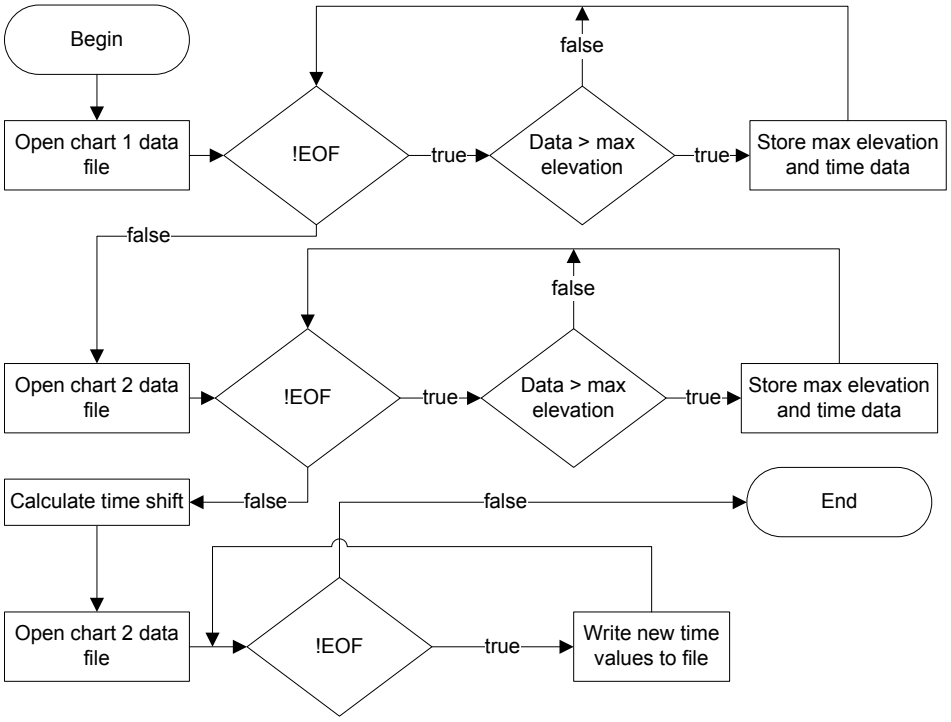


Figure 5.17: shift_sat_to_aircraft function flow diagram

has been entered. If the coordinate is valid, the function converts the text input to doubles and stores the converted data.

### Edit_elevationExit (FU5 GS_Interface)

The Edit_elevationExit function is invoked each time the user changes the Max elevation text box from the ground station window. See Figure 5.21. The function then proceeds to check that a valid elevation angle has been entered. If a valid elevation angle has been entered, the function calculates the area of the satellite elevation-time graph by calling the CalArea (FU3) function, with the maximum elevation angle as the parameter.

### Edit_speedExit (FU5 GS_Interface)

The Edit_speedExit function is invoked each time the user changes the Velocity text box from the ground station window. See Figure 5.21. The function then proceeds to check that a valid velocity value has been entered. If a valid velocity value has been entered the function directs the program flow to calcu-

late the values for aircraft altitude above the ground station and the minimum ground station to nadir distance. These values are displayed to the user in the Altitude above GS and Distance from GS text boxes; refer to Figure 5.21.

The function starts by calling the bestHV (FU3) function to calculate the appropriate aircraft altitude. Then the GStoNadir (FU3) and Cal_DGround (FU3) functions are called to calculate the aircraft minimum ground station to nadir distance and the distance of the flight path on the ground. After these calculations, the function calls the GStoNadir (FU3) and Cal_DGround (FU3) functions again to calculate these parameters for the satellite.

### GraphsClick (FU5 GS_Interface)

The GraphsClick function is called when the user clicks the Graphs button of Figure 5.21. The GraphsClick function invokes the GS graph window (Figure 5.22). The GS graph window displays the elevation and azimuth-time graphs of that ground station.

### Button_Cal_AzimuthClick (FU5 Main)

The Button_Cal_AzimuthClick function calls the CalAzimuth (FU3) function, that calculates the azimuth for the specified flight paths. The Button_Cal_AzimuthClick function is invoked by clicking on the Calculate Azimuth button. See Figure 5.20.

### Change scenario view (FU5 Main)

The following functions form part of FU5 and allow the user to change the viewing prospective of the scenario map by calling the appropriate function located in FU1:

**Button_downClick** Move the scenario map down.

**Button_upClick** Move the scenario map up.

**Button_leftClick** Move the scenario map to the left.

**Button_rightClick** Move the scenario map to the right.

**Button_zoomminClick** Zoom out.

**Button_zoomplusClick** Zoom in.

These function are invoked by buttons located on the main application window (Figure 5.20).

### Button_GS_1..12Click (FU5 Main)

Invokes the ground station interface form that allow the user to specify the ground station parameters.

### Cal_D (FU6 flightcalculations)

The Cal_D function calculates line of sight distance between two coordinate points. The coordinates of these two point are the input parameters to this function; the function then returns the result.

The function starts by converting the received LLA (latitude, longitude and altitude) coordinates to a Cartesian ECEF coordinate system. The distance between the two points is then calculated by using Pythagoras.

### Cal_E_A (FU6 flightcalculations)

The Cal_E_A function calculates the aircraft elevation and azimuth angles relative to a specific ground station. This function forms part of FU6 and calculates these angles in flight. The coordinates of the ground station are given as the input parameters to this function. The calculations implemented in this function to calculate the elevation and azimuth angles are presented in sections 2.6.7 and 2.6.8.

### setFlightTimeIntervalBegin (FU6 flightcalculations)

This function returns the current time of the Windows operating-system clock.

### getFlightTimeInterval (FU6 flightcalculations)

The getFlightTimeInterval function returns the time interval. The function is used to measure the time intervals as the aircraft flies past a ground station. The input parameter to this function is the start time of the time interval. The start time is the current time read from the Windows operating-system when the aircraft is at the start of the flight path past a ground station. The getFlightTimeInterval function then reads the current time from the Windows operating-system clock and calculates the time interval by subtraction.

#### OnFlightPath (FU6 flightcalculations)

The task of OnFlightPath function is to check if the aircraft is on a flight path past a ground station. If this is true the function proceeds to call the Cal_E_A function that handles the in flight elevation and azimuth calculations. The function is called each time a new telemetry packet is successfully received.

The OnFlightPath function uses the Cal_D function to calculate the direct line of sight distance between the aircraft and the nearest way-point. A flag is set if the distance is small enough to indicate that the aircraft is within the way-point. Thus the flag indicates that the aircraft is on the flight path past a ground station. The Cal_E_A function will then be called to calculate the in flight elevation and azimuth angles. The flag will be reset when the aircraft leaves the second way-point on the flight path and in this case the Cal_E_A function will not be called.

#### SwitchPayload (FU6 flightcalculations)

The SwitchPayload function indicates whether the payload should be switched on or off. As mentioned in this thesis, the payload should only be switched on when the aircraft is on the flight path past a ground station.

This function is called to evaluate the state of the payload each time a new telemetry packet is successfully received. The state of the payload is calculated with the help of a state machine. In the initial state the payload is switched off. When the aircraft reaches a way-point the function will proceed to the second state of the state machine that switches the payload on. The function will then remain in this state until the aircraft reaches the second way-point of the flight path, where the function will proceed to the third state. The task of the third state is to check for the case when the aircraft leaves the second way-point. If this occurs, the function proceeds to the initial state that switches the payload off.

### 5.1.7 Graphical User Interface

Figures 5.18 show a butterfly diagram of the functions that form part of the ASE and are invoked by the GUI.

Figure 5.19 shows the first window of the GUI that is displayed when the ASE application is launched. The layout of this window is as follows:

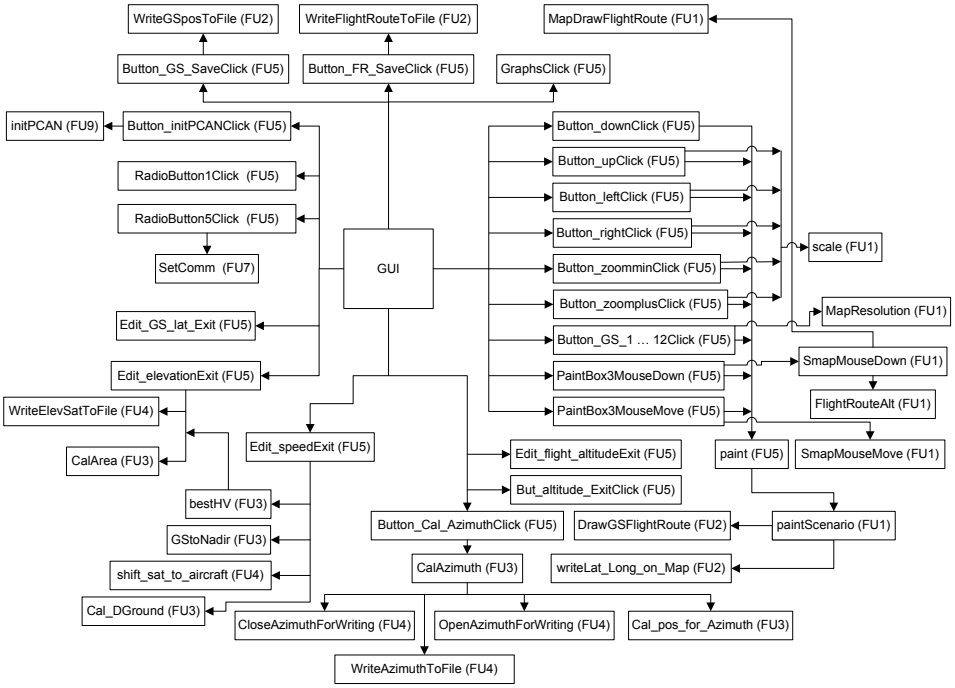


Figure 5.18: Butterfly diagram of functions invoked by the GUI

**Tabs** Enables the user to switch between the aircraft information and the flight route windows.

**Initialise peripheral device panel** This panel allows the user to initialise the ASE peripheral devices and set the radius of each the way-point. The serial interface to the aircraft avionics equipment and the payload are initialised by choosing the corresponding com port radio button. The CAN module is initialised by clicking on the *initialise PCAN* button.

**Aircraft information panel** Displays the aircraft information received from the aircraft avionics equipment.

Figure 5.20 shows a screenshot of the window invoked if the user chooses the flight route tab. The layout of this window is as follows:

**Scenario map** Provides a map of the scenario. Latitude and longitude coordinates are displayed at the edges of the map.

**Aircraft** This red dot represents the aircraft position. The position of the red dot will change as the aircraft moves.

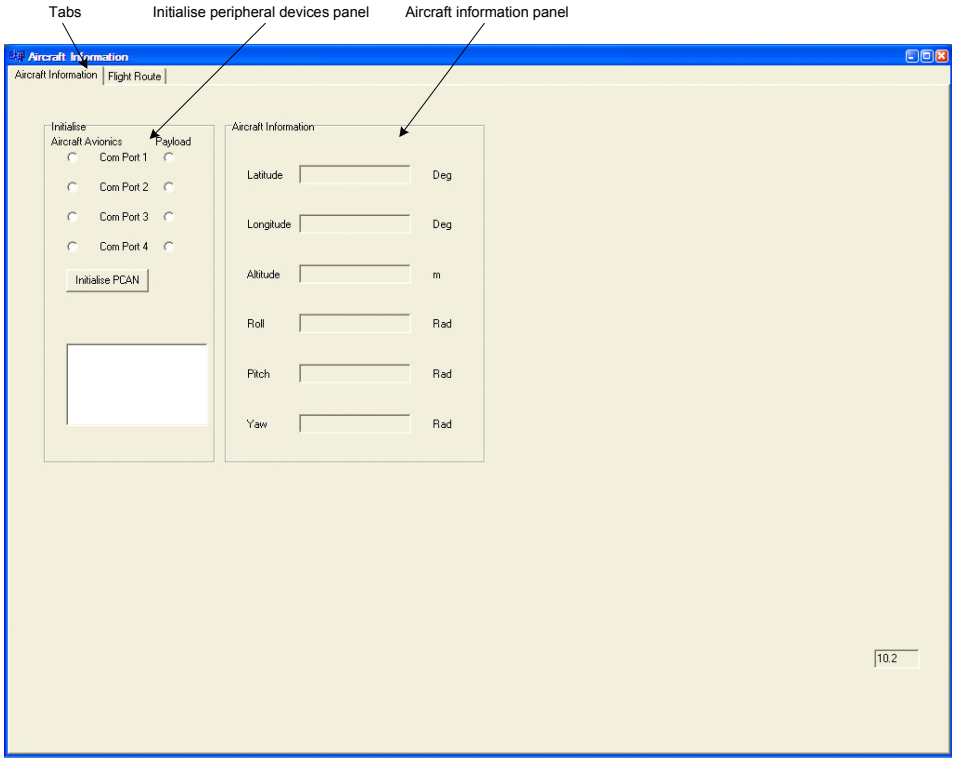


Figure 5.19: Screenshot of the aircraft information tab of the GUI

**Ground stations** The blue dots represent the ground station positions. If the user clicks on a ground station, the orientation of the flight path past that ground station can be changed.

**Flight path** The flight path.

**Way-points** The way-points are located at the start and end of every flight path past a ground station. The way-point on screen changes colour as the aircraft moves through it.

**Ground station panel** A window (Figure 5.21) that enables the user to set the ground station specifications is invoked if the user clicks on one of the ground station buttons located in the ground station panel. The radius of the sphere for the way-points located at the start and end of the flight path past a ground station can be changed in this panel by changing the value in the text box.

**Flight route panel** The flight route panel allows the user to specify the rest

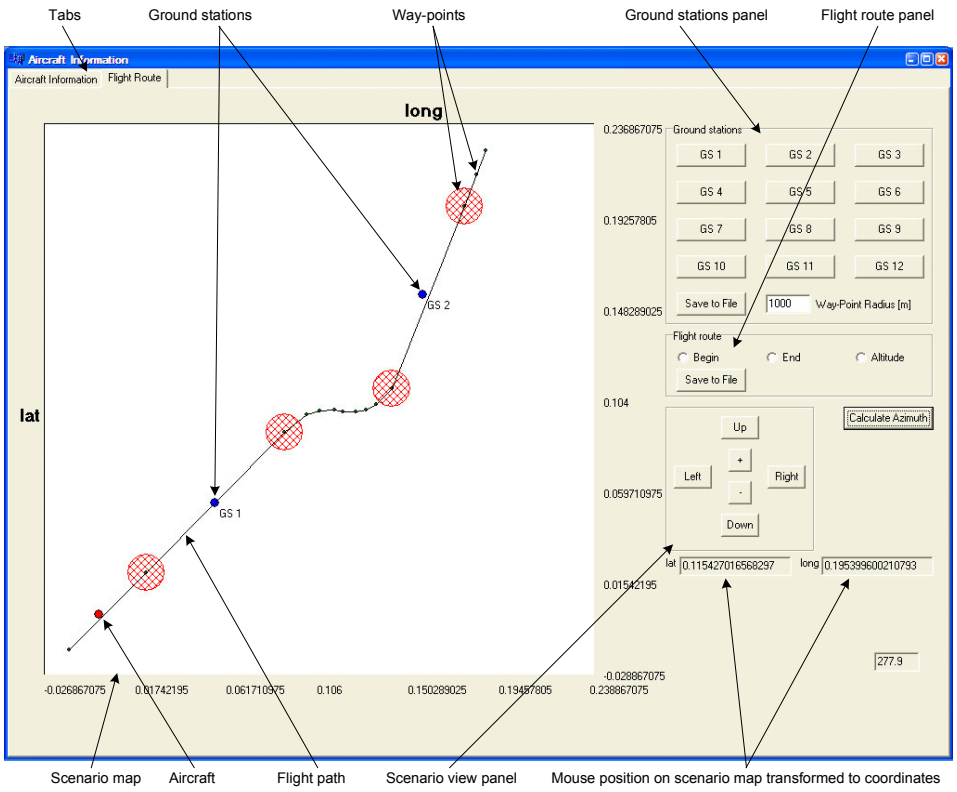


Figure 5.20: Screenshot of the flight route tab of the GUI

of the flight route, meaning that part of the flight route that is not between the two way-points that specify the flight path past a ground station. This part of the flight route is specified as follows: The user first selects the *begin* radio button, then proceeds to click on the scenario map to specify the latitude and longitude coordinates of the way-points. As the user specifies the way points, the application will draw the flight route. After all the way points have been specified, the *end* radio button must be selected. The user must then proceed to specify the altitude of the way-points. This is done by selecting the altitude radio button and clicking on the specified way-points. A window will then appear that allows the user to edit the altitude for that way point of the flight route.

**Mouse position on scenario map transformed to coordinates** The position of the mouse on the scenario map is transformed to latitude and longitude coordinates and displayed to the user by these two text boxes.

**Scenario view panel** The scenario view panel allows the user to move the

scenario map up, down, right, left and to zoom in and zoom out.

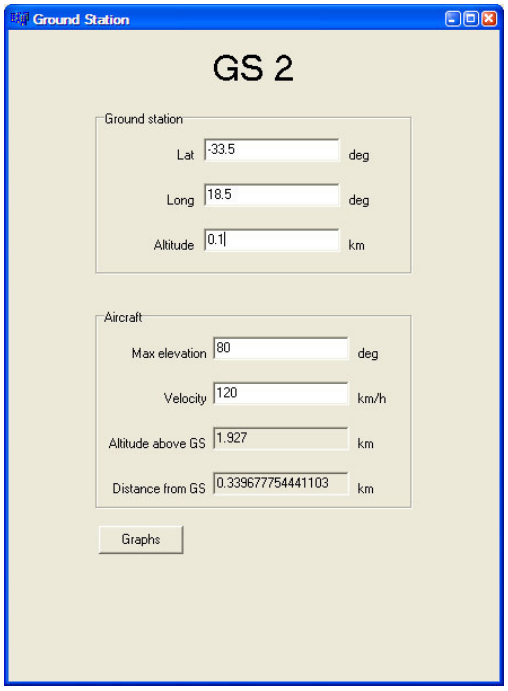


Figure 5.21: Screenshot of the ground station and aircraft information tab of the GUI

Figure 5.21 shows the GUI window invoked if the user chooses a *ground station* button from the ground station panel. The window allows the user first to specify the coordinates of the ground station. Thereafter the aircraft parameters are specified. The aircraft parameters specified by the user are the maximum elevation angles and aircraft velocity. The application then calculates and displays the altitude above the ground station and the shortest distance from the nadir point to the ground. The *graphs* button provided in this window open another window, if selected, that displays elevation and azimuth graphs for that ground station.

Figure 5.22 displays a elevation-time graph for a ground station. The graph displays calculated satellite elevation angles, calculated aircraft elevation angles and the flight aircraft elevation angles.

Figure 5.23 displays a azimuth-time graph for a ground station. This window is invoked by selecting the *azimuth* tab from the *graphs* window. The

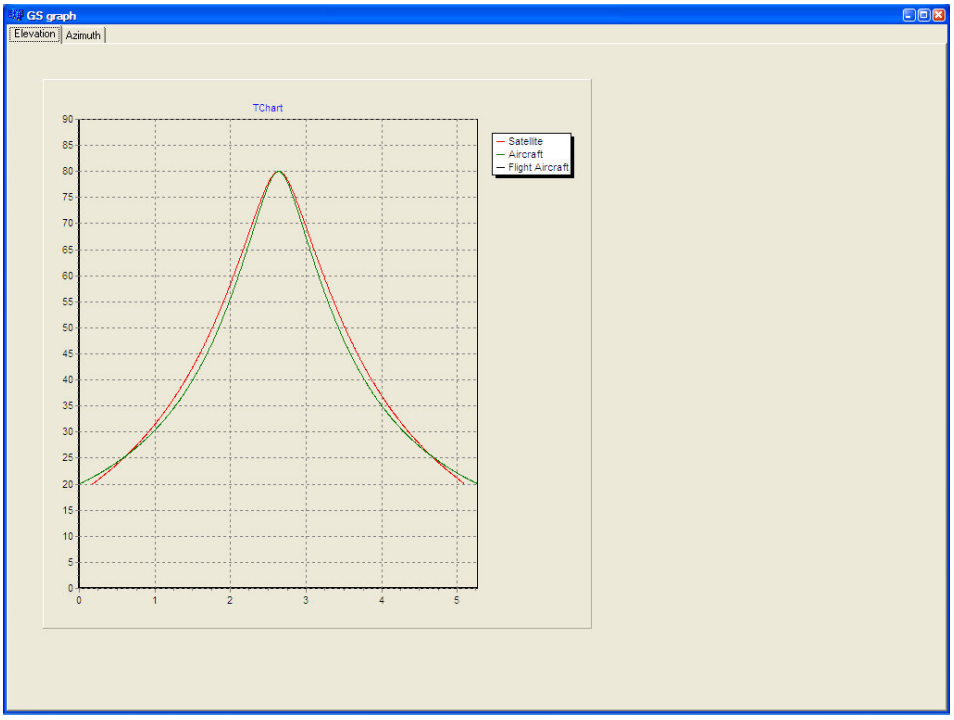


Figure 5.22: Screenshot of the elevation-time graph tab of the GUI

graph displays calculated satellite azimuth angles, calculated aircraft azimuth angles and the flight aircraft azimuth angles.

## 5.2 SAA Control

### 5.2.1 Virtual CAN Node (FU12)

In the following subsection the various functions the of virtual CAN node functional unit (FU12) are explained.

#### init_CAN

The SH4 has two CAN nodes connected to the C&DH bus and the ADCS bus. Each CAN node has a Cygnal processor with its own unique ID. A few node IDs are also assigned to each Cygnal processor. The Cygnal processor will let a CAN packet only through if the destination ID of the received CAN packet matches one of its own CAN node IDs.

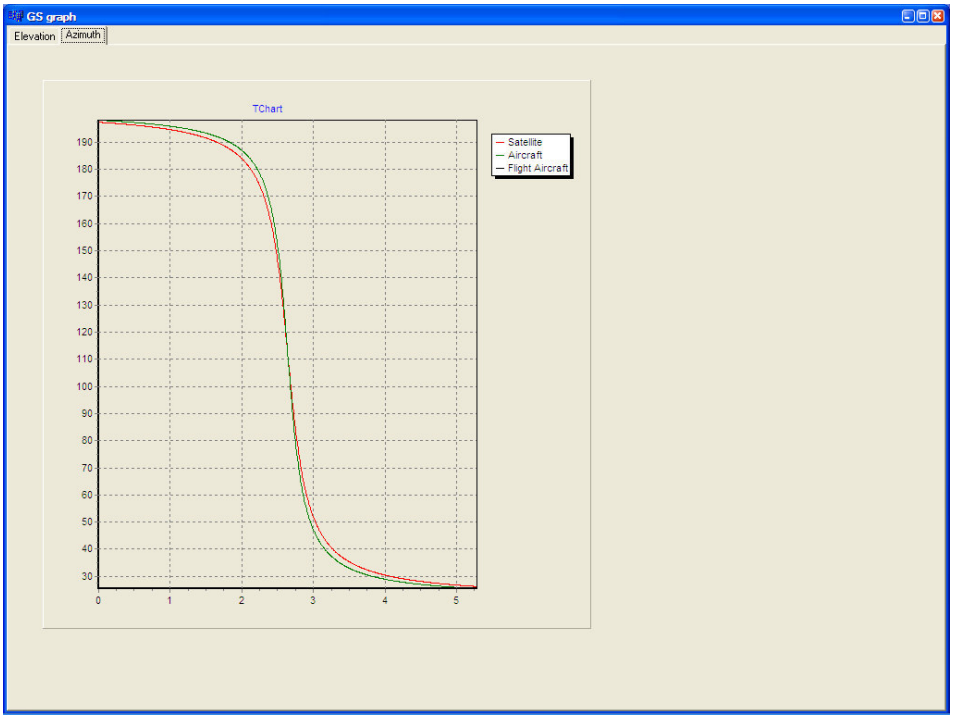


Figure 5.23: Screenshot of the azimuth-time tab of the GUI

As mentioned, the CAN driver implemented here creates a virtual CAN node that makes use of the existing CAN driver on the SH4. The CAN driver on the SH4 is initialised by the start up code of the SH4. The CAN driver for the virtual CAN node uses functions from the CCI libraries. It is initialised by a system call from these libraries, which opens a handle to the actual CAN driver. The system call initialises a unique CAN node ID by choosing a slot.

The command line of the SH4 on the experimental payload can be used to display the available CAN node IDs for a certain bus. Figure 5.24 illustrates this. On the experimental payload IDs 8 and 136 are assigned to bus Bus #0 and Bus #1 respectively. Bus #0 is connected to the C&DH bus and Bus #1 to the ADCS bus. The number before the RxPackets is the CAN node ID. By choosing a slot it is possible to choose a specific CAN node ID. This implies that if the system call chooses slot 1, then a CAN node ID of 10 is assigned to the virtual CAN node. On the experimental payload slot 0 is chosen by the SH4, thus all the CAN packets with a destination ID of 9 will be let through to the SH4. However, the CAN driver implemented here chooses slot 1, thus all the CAN packets with a destination ID of 10 will be let through to the

virtual CAN node.

```
# cat /dev/can/stats

CAN Statistics, v1.7.0.319

Bus  #0:   8, RxPackets = 895047, RxDropped = 0, TxPackets = 1798037, TxDropped = 0
Bus  #1: 136, RxPackets = 1, RxDropped = 0, TxPackets = 1, TxDropped = 0

Slot #0:   9, RxPackets = 4, RxDropped = 1, TxPackets = 7, TxDropped = 0, *
Slot #1:  10, RxPackets = 0, RxDropped = 0, TxPackets = 0, TxDropped = 0
Slot #2:  11, RxPackets = 0, RxDropped = 0, TxPackets = 0, TxDropped = 0
Slot #3: 137, RxPackets = 0, RxDropped = 0, TxPackets = 0, TxDropped = 0
Slot #4: 138, RxPackets = 0, RxDropped = 0, TxPackets = 0, TxDropped = 0
Slot #5: 139, RxPackets = 0, RxDropped = 0, TxPackets = 0, TxDropped = 0
SlotNet:   9, RxPackets = 894830, RxDropped = 212, TxPackets = 1798031, TxDropped = 0, *
#
```

Figure 5.24: SH4 command line

### TcmHandler

The TcmHandler function is invoked by the virtual CAN node each time a telecommand packet is received. The function decodes the telecommand message by analysing the message ID. The message IDs are analysed by a switch statement. The telecommand handled by this function sets a flag. If the flag is set to 0, the loop in the program flow of the main thread in FU13 will stop. This will enable FU13 to close all the initialised systems and the SAA software to exit.

### Request_sat_data

The Request_sat_data function requests data from the ASE. Figure 5.25 shows a flow diagram of this function. The requested data contains the attitude as well as the coordinates of the aircraft. The ASE is connected via a CAN bus to the OBC that contains the control software. A virtual node is used by the control software application on the OBC to interface with the CAN bus. The *request telemetry* function from the CCI libraries of the virtual node is used to send the telemetry request on the CAN bus. The function specifies a frame ID, which makes is it possible to specify specific data. Table 5.1 shows the grouping of the telemetry frames. After a telemetry request the system call waits for a response from the ASE, which contains the requested data. If no response is received after a specified amount of time, the call will exit and the Request_sat_data function return with a fail message. If a reply is received, the received data is first stored in a temporary location. After

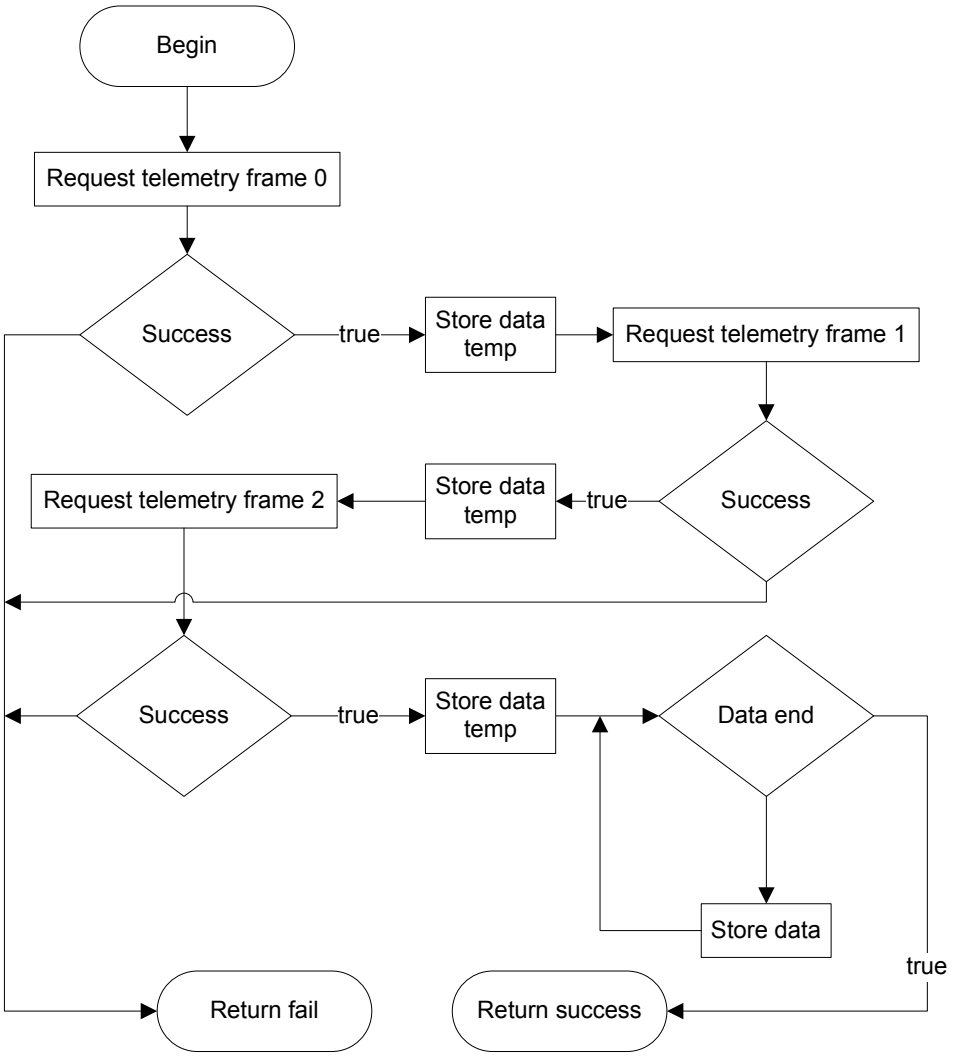


Figure 5.25: Request data from ASE flow diagram

all three telemetry frames have been received the function loops through the data and stores it in a data structure. The function then exits with a success message.

### Sat_data_Gain

The data received from the ASE needs to be multiplied by a constant to give the correct value. The data contains the attitude and the satellite coordinates. Table 5.2 shows the gain required by the corresponding data value and its unit.

| Telemetry frame number | Telemetry frame description | Size (Bytes) | Frame bytes | Format |
|---|---|---|---|---|
| 0x00 | Latitude | 4 | 0 - 3 | Signed 32 bit Int |
| | Longitude | 4 | 4 - 7 | Signed 32 bit Int |
| 0x01 | Altitude | 4 | 0 - 3 | Signed 32 bit Int |
| 0x02 | Roll | 2 | 0 - 1 | Signed 16 bit Int |
| | Pitch | 2 | 2 - 3 | Signed 16 bit Int |
| | Yaw | 2 | 4 - 5 | Signed 16 bit Int |

Table 5.1: Grouping of telemetry frames

| Data | Gain | Unit |
|---|---|---|
| Latitude | $10^{-7}$ | Degrees |
| Longitude | $10^{-7}$ | Degrees |
| Altitude | $10^{-7}$ | Degrees |
| Roll | $\frac{2\pi}{2^{15}}$ | Radians |
| Pitch | $\frac{2\pi}{2^{15}}$ | Radians |
| Yaw | $\frac{2\pi}{2^{15}}$ | Radians |

Table 5.2: Gain required by each data value and its unit

### 5.2.2 SAA Interface (FU17)

The SAA control module controls the steerable antenna by transmitting commands to the SAA via a serial port. The SAA module has a FPGA that is responsible for all the systems on board. It is with this FPGA that the SAA interface communicates.

#### dbs_init

The dbs_init function initialises the serial port used to control the SAA. Standard POSIX functions are used to control the drivers on the QNX operating system. By controlling the serial driver (devc_sersci) on the QNX operating system it is possible to control the serial port. There are two serial ports available on the SH4 board. Port 1 is located at /dev/ser1 and port 2 at /dev/ser2. An application can open one of these ports and read or write from it. The data written to it will be transmitted from the serial port.

The dbs_init function uses the *POSIX open* function to gain access to the

serial device driver. It is also important to set the baud rate to 38400 and disable the flow control. This allows an application on QNX to communicate with the SAA on the serial port.

### dbs_initSAA

The dbs_initSAA function initialises the SAA. Figure 5.26 shows a flow diagram of this process. The function first synchronises the FPGA. This is achieved by calling the dbs_sync_commandchain function. The synchronisation, by flushing the FPGA input buffer, makes sure that the FPGA is ready to receive new data. The buffer is flushed by writing 0s to the FPGA until the FPGA responds by writing a 0 back. The FPGA will reply with a conformation byte of 0 upon the reception of a few 0s. If the FPGA is successfully synchronised, the function will then proceed to set up the PLL and reset all the FPGA components on board the SAA. The PLL provides for the reference frequency to the RF elements. The PLL was set up according to the specifications provided by KU Leuven.

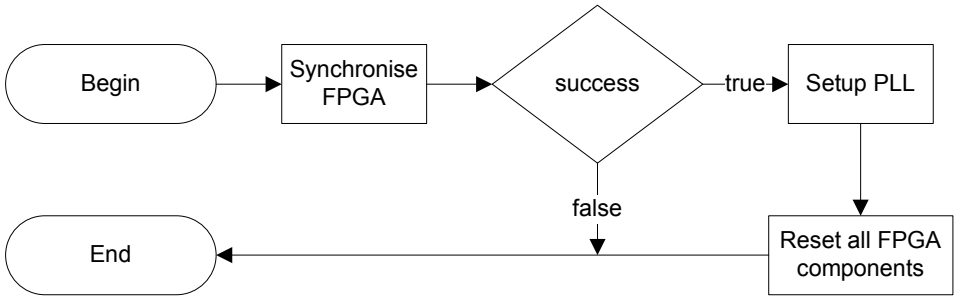


Figure 5.26: Initialise SAA function flow diagram

### dbs_steer_array_to

This function steers the SAA according to the steering angles calculated. The SAA is steered by controlling the phase shifter of each element. The dbs_steer_array_to function calculates and sets the multiplier values for each phase shifter. The phase shift of the signal from an element is controlled by changing the multiplier values of the element in the array. Each element has 4 multipliers. The signal received before the phase shift contains an in phase

I and quadrature Q component. Rotating the signal by $\triangle\theta$ using a matrix rotation is the signal phase shifted by $\triangle\theta$. The values in the matrix are the 4 values of the multipliers. Refer back to section 2.9 for an illustration of these components.

The $\triangle\theta$ value is the amount the signal of an element has to be phase shifted by. The array elements are spaced a specified wavelength from each other in the antenna array. A signal received by the antenna array at an angle will arrive at each element at a slightly different time. Thus each element has to be phase shifted by a different amount.

The formula to calculate the phase shift $\triangle\theta$ was provided by KU Leuven. The calculation takes into account the position of each element in the array by compensating for the phase imbalance between the elements. The matrix values containing the phase imbalance are also supplied by KU Leuven.

$$
\begin{aligned}
\textit{element } 1: & \begin{bmatrix} \cos(\triangle\theta_1) & \sin(\triangle\theta_1) \\ -\sin(\triangle\theta_1) & \cos(\triangle\theta_1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \\
\textit{element } 2: & \begin{bmatrix} \cos(\triangle\theta_2) & \sin(\triangle\theta_2) \\ -\sin(\triangle\theta_2) & \cos(\triangle\theta_2) \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix} \\
\vdots \qquad & \qquad\qquad \vdots \qquad\qquad = \qquad \vdots \\
\textit{element } x: & \begin{bmatrix} \cos(\triangle\theta_x) & \sin(\triangle\theta_x) \\ -\sin(\triangle\theta_x) & \cos(\triangle\theta_x) \end{bmatrix} = \begin{bmatrix} (x-1)4 & (x-1)4+1 \\ (x-1)4+2 & (x-1)4+3 \end{bmatrix}
\end{aligned}
$$

Figure 5.27: Numbering of the multipliers

The ID number for each multiplier is assigned in the following way: Each element has four multipliers and a unique matrix. The matrix is used to calculate the phase shift of the received signal. The values in the matrix correspond to the multiplier values. The numbering system of the multipliers is illustrated by Figure 5.27.

Figure 5.28 shows the flow diagram of the implementation of this function. The function first calculates the multiplier values for the first element. The first element is the reference element and all the other elements are calibrated with reference to this element. The function will then proceed to calculate the phase shift and multiplier values of all the remaining elements. After all the multiplier values have been calculated, the function sets the multiplier by

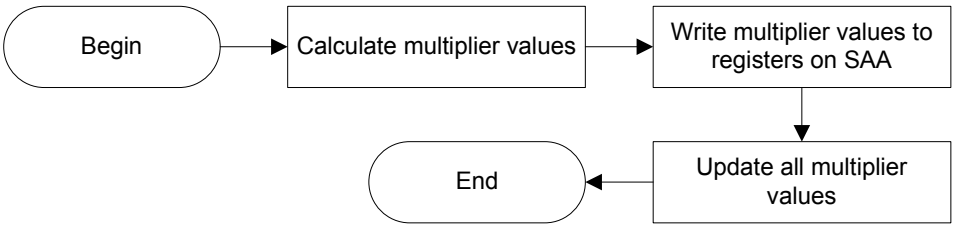


Figure 5.28: The dbs_steer_array_to function flow diagram

two commands. The first command writes the calculated multiplier values to a register in the SAA. Each multiplier on the SAA has a register that acts as a buffer. The second command updates all the multipliers with the value contained in the multipliers corresponding register.

The set *multiplier value* command sends a value to a specific multiplier register on the SAA. Table 5.3 depicts the format of this command package.

| Byte | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Description | Command ID | Multiplier ID | Data (LSB) | Data (MSB) | For future use |
| Format | 01 | | | | XX |

Table 5.3: *Set multiplier value* command format

The main parts of the *set multiplier value* command package are summarised below.

- The command ID byte identifies the command type.
- The multiplier ID specifies the specific multiplier intended for this data.
- The data sent is a signed 12-bit value.

If the *update all multipliers* command is received by the SAA, it will proceed to update all the multipliers with the values contained in the multiplier registers. Table 5.4 shows the command format.

### 5.2.3 Scheduling Interface (FU14)

This functional unit receives the coordinates of the ground station to which the SAA has to steer from the scheduling software (FU15). The scheduling interface is implemented by a thread that is responsible for the acquiring of

| Byte | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Description | Command ID | | | | |
| Format | 02 | XX | XX | XX | XX |

Table 5.4: *Update all multipliers* command format

the identity of the target ground station from the scheduling software. Figure 5.29 illustrates the communication sequence between the scheduling interface and the scheduling software. The scheduling interface sends a request to the scheduling software. The coordinates of the ground station to which the SAA has to be directed are requested from the scheduling software. The scheduling interface will then block until a message with the requested data is received. The scheduling software will only send the reply containing the coordinates at the time when the SAA needs to be directed to that ground station. Thus the scheduling software sends the coordinates of the target ground station only once the SAA has to be directed to a new location. This communication between the scheduling interface and scheduling software is implemented with the help of inter process communication (IPC).

IPC is the preferred method of message passing in QNX. IPC allows communication and data transfer between processes. The communication between processes can be thought of as a server-client relationship. The server creates a channel by creating an attached point. A client then connects to the server by connecting to the attached point. This implies that a channel is opened between the server and client once the client has successfully connected to the attached point created by the server.

The communication between the server and client can be described as follows: The server first performs a *message receive*. The *msgReceive* function will then block until a message is received from the client. If a message is received from the client, the server will send a reply message to the client. Note that the *msgSend* function in the client will block once a message is sent to the server, but only until a reply message is received from the server. The *msgSend* and *msgReceived* functions are provided by QNX and enable data transfer between processes. Figure 5.30 illustrates this communication protocol between the server and client. The scheduling software acted as the server and the scheduling interface as the client in the implementation of the IPC

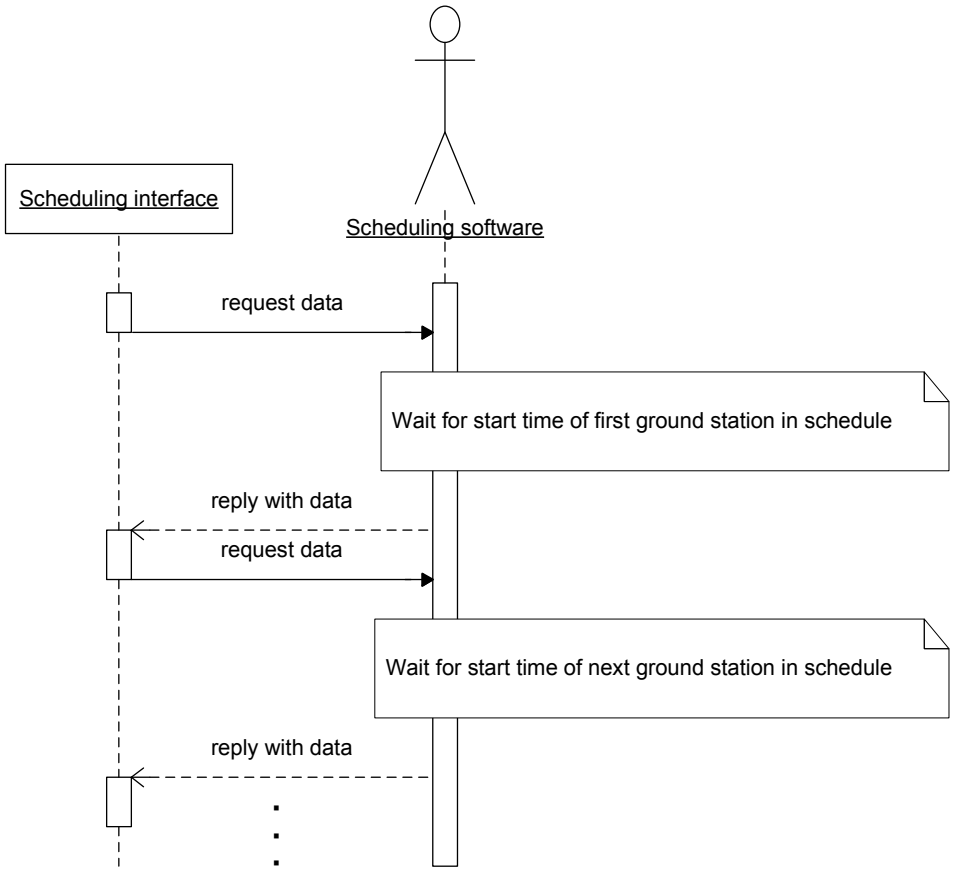


Figure 5.29: Scheduling interface sequence diagram

method in this project. The functions implemented to enable the software interface will be described next.

### create_threads

The create_threads function initialises the scheduling interface. The scheduling interface is initialised by creating a thread. The thread created will request the target ground station coordinates from the scheduling software. The thread created will be discussed next.

### Read_gs_pos_thread

This thread first initialises the connection to the scheduling software. The thread then sends a message to the scheduling software requesting the target ground station coordinates. After the request is sent the thread will block

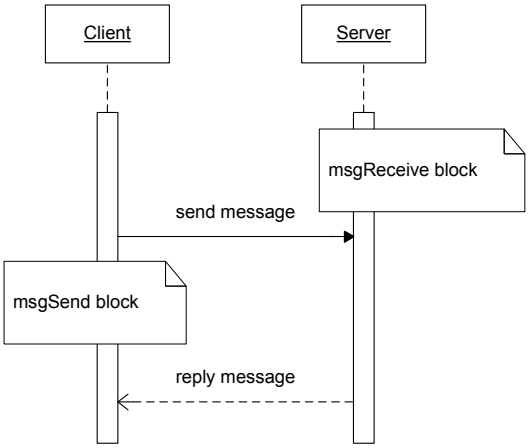


Figure 5.30: Client-server sequence diagram

until a reply is received. The reply contains the requested data. The thread will then proceed to update the target ground station coordinates. Figure 5.31 shows the flow diagram of this thread.

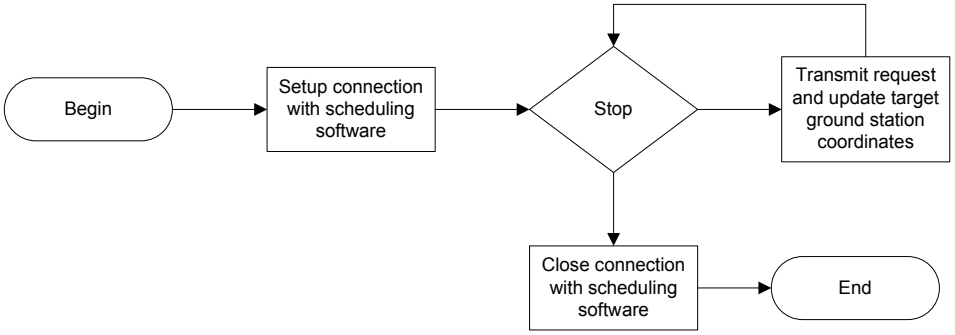


Figure 5.31: Flow diagram of the scheduling interface thread

### init_scheduling_handler

init_scheduling_handler function sets up a connection with scheduling software by connecting to the attached pointed created by the scheduling software. The function uses the name_open function provided by QNX to open this connection. The ID of the channel is then returned by the name_open function if a successful connection was achieved.

**transmit_to_scheduling**

The transmit_to_scheduling function requests the target coordinates from the scheduling software and updates the target coordinates once a reply is received from the scheduling software. The function requests data by performing a *msgSend*, and the *msgSend* function will block until the reply is received from the scheduling software. The data that was requested is contained in the reply. Note that mutex locks are used when this function updates the target coordinates. These locks are necessary to ensure data integrity when data is accessed by a different thread . In this case the scheduling interface is one thread and the rest of the SAA control software is another thread. Figure 5.32 shows the flow diagram of this implementation.

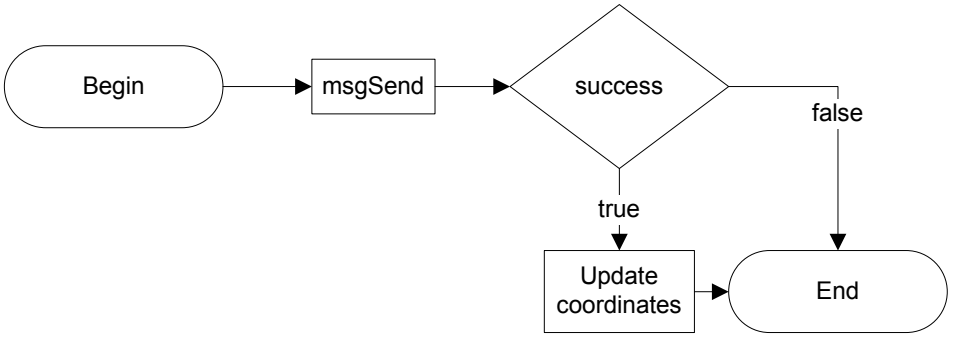


Figure 5.32: Flow diagram of transmit_to_scheduling function

### 5.2.4 Calculate Steering Angles (FU16)

This section will describe the basic strategy to calculate the $\phi$ (phi) and $\theta$ (theta) angles. These angles will enable the airborne object to track the specified ground station. These angles are defined in section 4.6.4 in the system design phase of the project.

The coordinates (LLA), specified as latitude, longitude and altitude, of both the airborne object and ground station are known. The first step is to convert these coordinates from the LLA system to the ECEF frame, by making use of the World Geodetic System 1984 standard (WGS-84)[29]. The results

can be written in the following way:

$$\vec{R}_{Sat} = \begin{bmatrix} X_{Sat} & Y_{Sat} & Z_{Sat} \end{bmatrix}^T and \quad (5.2.1)$$

$$\vec{R}_{GS} = \begin{bmatrix} X_{GS} & Y_{GS} & Z_{GS} \end{bmatrix}^T \quad (5.2.2)$$

The airborne object to ground station position vector is then calculated as

$$\vec{R}_{pos} = \vec{R}_{GS} - \vec{R}_{Sat} \quad (5.2.3)$$

The position vector can now be converted from the ECEF frame to the NED frame situated at the airborne object.

$$\vec{B} = \mathbf{K} \cdot \vec{R}_{pos} \quad (5.2.4)$$

where $\mathbf{K}$ is the transformation matrix [6].

It is necessary to take the attitude of the object into account. This can be done by describing the object in terms of its pitch, roll and yaw angles. By using an Euler 123 rotation [30] to transform the position vector from the NED frame to the body frame of the object, it is possible to describe the position vector from the object to the ground station in terms of the orientation of the object. The $\theta$ and $\phi$ angles can be calculated after the attitude of the object has been taken into account.

$$\vec{W} = \mathbf{A} \cdot \vec{B} \quad (5.2.5)$$

The $\theta$ and $\phi$ angles are relative to the body frame of the object. These angles direct the position vector to the target or ground station and can be calculated by means of simple trigonometry. Figure 5.33 defines the parameters used to calculate these angles.

The theta $\theta$ angle is calculated as:

$$t = \sqrt{W_x^2 + W_y^2} \quad (5.2.6)$$

$$\theta = \frac{\pi}{2} - \arctan\left(\frac{W_z}{t}\right) \quad (5.2.7)$$

$$= \frac{\pi}{2} - \arctan\left(\frac{W_z}{\sqrt{W_x^2 + W_y^2}}\right) \quad (5.2.8)$$

The $\phi$ angle is calculated as:

$$\phi = \arctan\left(\frac{W_y}{W_x}\right) \quad (5.2.9)$$

The control algorithm is summarised below.

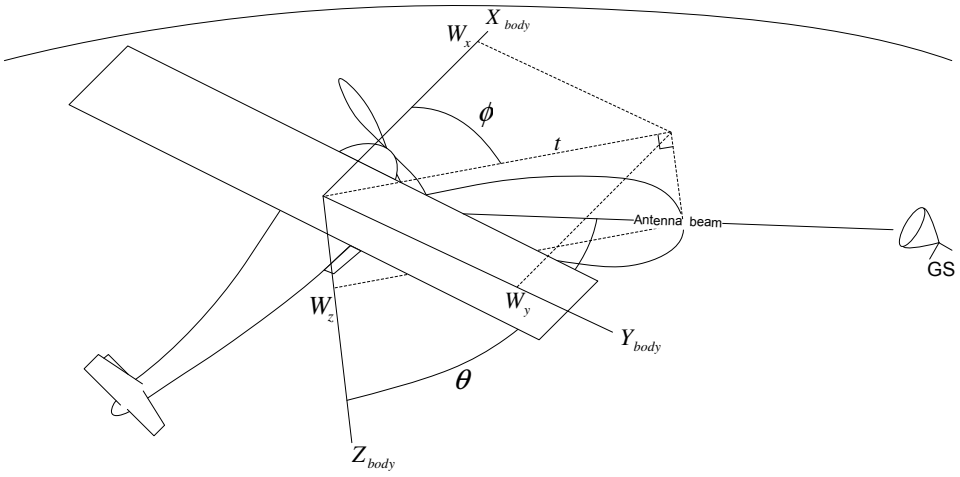


Figure 5.33: Defining parameters used to calculate the $\theta$ and $\phi$ angles.

- Transform the latitude, longitude and altitude coordinates for the object and ground station to the ECEF frame.
- Calculate the position vector from the object to the ground station.
- Transform the position vector from the ECEF frame to the NED frame.
- Transform the position vector from the NED frame to the body frame of the object.
- Calculate the $\theta$ and $\phi$ angles.

### 5.2.5 Control Unit (FU13)

The control unit controls the functional units and data flow of the SAA Control module. The control unit uses the functional units to enable the SAA control software to control the SAA. The control software has to direct the SAA beam in the direction of the specified ground station. The antenna beam can be directed by making use of the satellite and ground station coordinates as well as the attitude of the satellite. Figure 5.34 shows a flow diagram of the implemented control unit.

The virtual CAN interface, serial port to the SAA and SAA are first initialised by the control unit. Then the thread for the scheduling interface (FU14) is created. See section 5.2.3 for a more detailed description of the scheduling interface and the initialisation of the this thread.

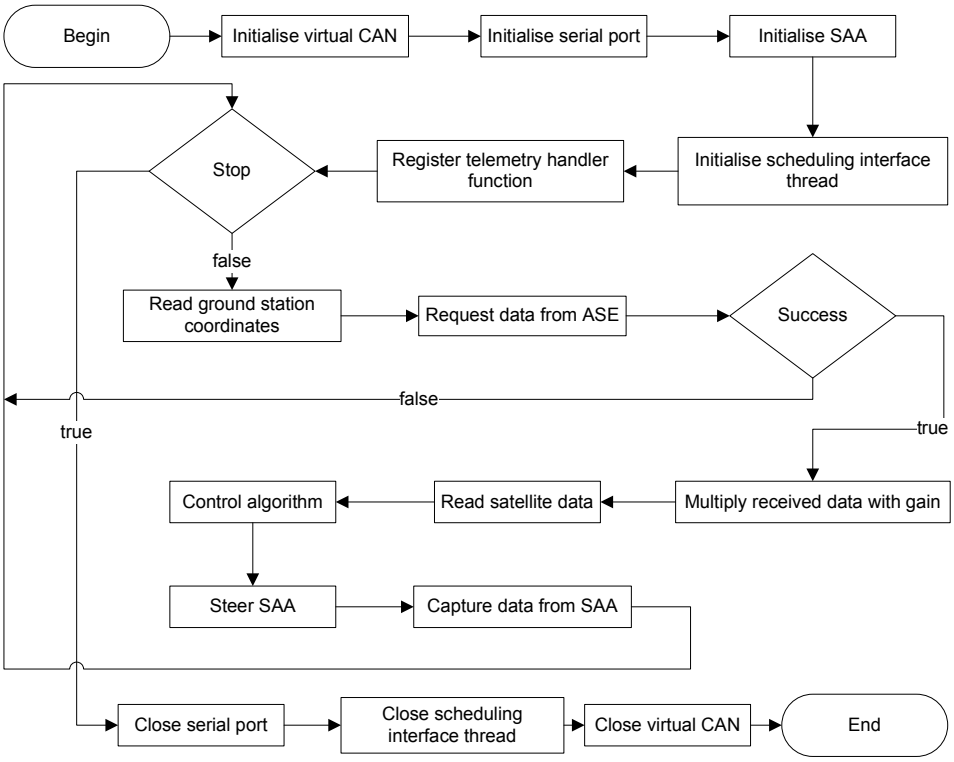


Figure 5.34: Control unit flow diagram

The function invoked each time a telemetry request is received by the virtual CAN node, must first be registered. The function TcmHandler is registered by the control unit. As mentioned in section 5.2.1, the TcmHandler function sets a flag. The flag indicates whether the program flow should stop and exit or continue. The ground station coordinates are read next if the flag indicates that the program should continue.

The ground station coordinates are obtained by the scheduling interface (FU14) from the scheduling software (FU15). A mutex lock is implemented to read the coordinates from the scheduling interface (FU14). Because the data coordinates are accessed by both threads, the mutex lock prevents both threads from accessing the data at the same time. The data stored at the same location can become corrupted if the threads access the data at the same time. The mutex lock works on the following principal. The mutex lock blocks and prevents access to the specified data until other threads have finished with the data and vise versa.

After the target ground station coordinates have been read, the next step is

to obtain the satellite data from the ASE. This data contains the coordinates and attitude of the satellite. The data is obtained by controlling the virtual CAN node functional unit (FU12). See section 5.2.1 for more detail of the functional unit. The functional unit requests this data from the ASE. If the data is successfully received, the functional unit stores the received data in a data structure and returns a reply indicating success.

The received data must then be multiplied by a specific gain value. After the multiplication the received data will contain the correct value in the unit specified. Table 5.2 shows the required gain values. The control unit implements this multiplication by using the Sat_data_Gain function of FU12. See section 5.2.1 for a description of this function.

The control algorithm calculates the $\theta$ and $\phi$ angles required to steer the antenna beam. The control algorithm is located in the calculate steering angles functional unit (FU16).

After the steering angles have been calculated, the control unit steers the SAA to the specified ground station by controlling the SAA. The function used to control the SAA is housed in FU17.

The next step in the control flow of the control unit is not mandatory, but can, however, be performed if desired. This is the capturing of the digital I and Q signals for each element and the summed I and Q signal after the phase shift on the SAA. This is achieved by writing a command to the SAA. The SAA will then reply with the captured data. The captured data is then saved on the OBC.

If the TcmHandler function sets the flag to stop the program flow will exit the main loop of the control unit. The control unit will then proceed to close all initialised peripheral devices and threads, thereafter exiting.

## 5.3 Conclusion

This chapter discussed the successful practical implementation of the systems design. In the subsequent chapter, the evaluation procedure and results will be presented.

# Chapter 6

# System Evaluation

## 6.1 Evaluation Approach

This chapter describes the evaluation and testing of the system performance. The evaluation approach first analysed two main system components, ie. the ASE and the SAA control modules, individually. The evaluation then proceeded to the evaluation of the various system interfaces. Thereafter, the overall system performance is evaluated in two different ways, first with an aircraft avionics equipment emulator, which was specifically developed for the evaluation of this system. The emulator serves as a substitute for an actual aircraft. Secondly, performance is evaluated with an aircraft simulator. The aircraft simulator closely models the light aircraft for which this system was developed. The aircraft simulator has input interfaces, such as a joystick, that contribute unknown factors such as pilot skill to the system evaluation. The chapter, however. starts by discussing the software and equipment used in the evaluation of the system.

## 6.2 Test Software and Equipment

The tests were performed with the following software/equipment.

- PCAN-View software application.
- Aircraft avionics equipment emulator (AAEE).
- Aircraft simulator.

### PCAN-View

The PCAN-View application was provided by PEAK-System. This software works in conjunction with the PCAN-USB module. The application allows the user to monitor the CAN bus and to send and receive CAN messages. The application was especially helpful in the development and testing of the CAN interface.

### Aircraft Avionics Equipment Emulator

The AAEE software was specifically developed in the course of the research for this project. The software provides realistic aircraft avionics data, in the correct protocol, to the system, thereby providing a means to test the system without an actual flight test.

The user can specify a chosen flight path by specifying the flight start-and-end coordinates as well as aircraft parameters such as velocity, altitude and attitude. The AAEE software uses this information to provide aircraft avionics data to simulate an aircraft flight. Figure 6.1 shows a screen shot of this application.

### Aircraft Simulator

The aircraft simulator is a standalone hardware based aircraft flight simulator. The simulator is used for training and automatic pilot development. Input devices such as a joystick control the simulated aircraft. Because the simulator closely models the light aircraft this system was developed for, the simulation will react in much the same way as the actual aircraft in flight. It was because of this extra dimension that the simulator brings to the evaluation, without an actual flight test, that this simulator was first used to evaluate the system. Figure 6.2 shows a screen shot of the GUI of the simulator.

## 6.3 ASE Tests

The tests described in the following subsections were performed to test the integrity of the ASE software.

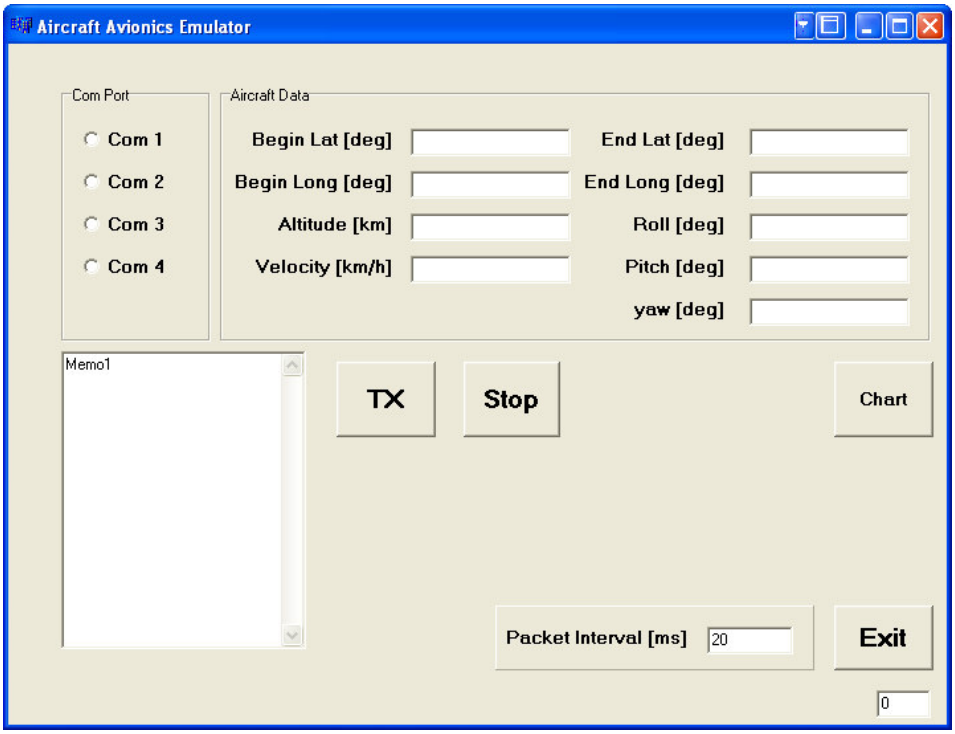


Figure 6.1: Screen shot of the aircraft avionics equipment emulator software GUI

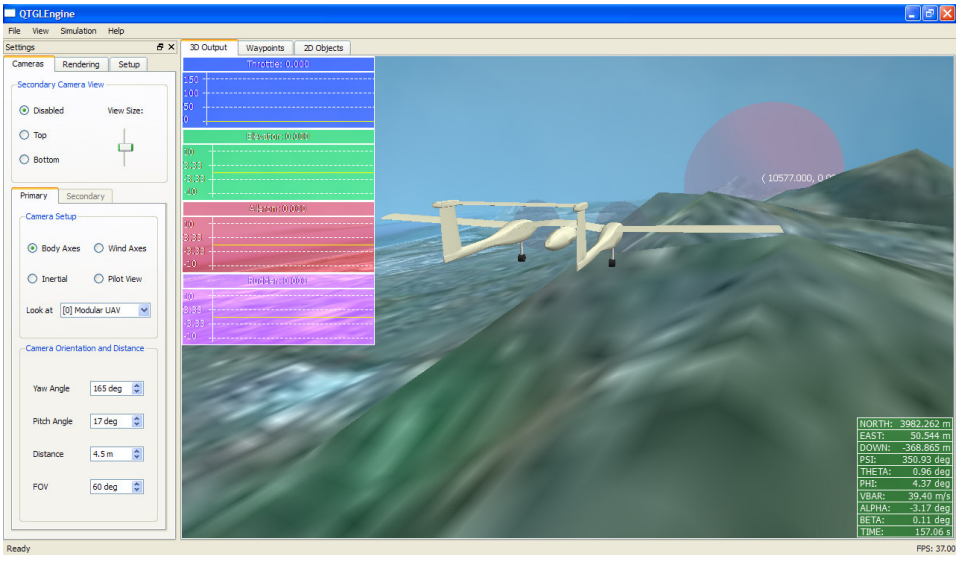


Figure 6.2: Screen shot of the aircraft simulator software GUI

### 6.3.1 Placement of Ground Stations on Flight Route Map

#### Objective

This experiment verifies that ground stations are placed at the correct coordinates on the flight route map.

#### Protocol

1. Run the aircraft satellite emulator application.
2. Generate a number of ground stations at different locations. This is done by using the ground station button of the GUI and specifying the coordinates.
3. Evaluate the placement of the ground stations on the flight route map generated by visually inspecting the flight route map and the generated file that saves the location of the ground stations.

#### Results

The placement of the ground stations on the flight route map concurred with the specified ground station coordinates.

### 6.3.2 Aircraft Altitude above Ground Station

#### Objective

The objective of this experiment is to verify that the correct altitude above a ground station is calculated by the ASE application. The results are verified by comparing the data generated to data produced by a Matlab script. The Matlab script was used in the emulation strategy phase of the project to calculate the parameters of an airborne object that will simulate a satellite pass.

#### Protocol

1. Change the input source for the maximum elevation angle and aircraft velocity from the GUI to a script.

2. Set up the script so that it generates various velocity values at different maximum elevation angles. Maximum elevation angles between 20 to 90 degrees at 5 degree intervals and velocity values between 60 to 300 km/h at 20 km/h intervals are generated by the script.

3. Change the output destination of the calculations from the GUI to a data file.

4. Run the ASE application.

5. Evaluate data generated.

#### Results

The data produced by the application was found to correspond to data generated by the Matlab script.

#### Discussion

The experiment shows that the application generates the correct altitude above ground station for a specific flight route.

### 6.3.3 Distance from Ground Station

#### Objective

The objective of this experiment is to verify that the correct distance from a ground station is calculated by the ASE application. The results are verified by comparing the generated data to data produced by a Matlab script. The Matlab script was used in the emulation strategy phase of the project to calculate the parameters of an airborne object that simulates a satellite pass.

#### Protocol

1. Change the input source for the maximum elevation angle and aircraft velocity from the GUI to a script.

2. Set up the script to generate various velocity values at different maximum elevation angles. The values for the maximum elevation angles ranged

from 20 to 90 degrees at 5 degree intervals and those for the velocity between 60 and 300 km/h at 20 km/h intervals.

3. Change the output destination of the calculations from the GUI to a data file.
4. Run the ASE application.
5. Evaluate generated data.

#### Results

The data produced by the application was found to correspond to data generated by the Matlab script.

#### Discussion

The experiment shows that the application generates the correct values for distance from ground station for a specific flight route.

### 6.3.4 Mapping Distance from Ground Station

#### Objective

This test verifies that the calculated distance from ground station is correctly mapped onto the flight route map.

#### Protocol

1. Run the ASE application.
2. Generate a ground station with a flight route.
3. Place flight route so that the start and end way-points of the flight route have the same longitude coordinate.
4. Read the latitude coordinates in degrees of the start and end way-points of the flight route from the flight route map. Then calculate the difference by subtracting the values read from each other.
5. Convert the calculated value in degrees to radials.

6. Calculate the distance by multiplying the converted radials by the summation of the ground station altitude and the average radius of the earth.

7. Compare the calculated distance with the distance from ground station calculated by the application.

#### Results

This experiment was repeated a few times for various maximum elevation angles at different ground station altitudes. The distance read from the map was then compared to the value calculated by the application. The distance read concurred with the values calculated in each case.

### 6.3.5 Mapping Flight Route Distance

#### Objective

The objective of this test is to verify that the distance of the flight route past a ground station is correctly mapped on to the flight route map. This is achieved by reading the coordinates of the start and end point of the flight route from the scenario map. The difference in degrees between the two points are then compared to the results generated by a Matlab script that calculates the degrees between the start and end point of a flight route.

#### Protocol

1. Run the ASE application.

2. Generate a ground station with a flight route.

3. Place flight route so that the start and end way-points of the flight route have the same longitude coordinate.

4. Read the latitude coordinates in degrees of the start and end way-points of the flight route from the flight route map. Then calculate the difference by subtracting the values read from each other.

5. Compare the data read to data calculated by the Matlab script.

#### Results

This experiment was repeated a few times for various maximum elevation angles at different aircraft speeds. The degrees between the start and end point of the flight route on the flight route map concurred with the data calculated by the Matlab script.

### 6.3.6 Elevation and Azimuth Graphs

#### Objective

The objective of this experiment is to evaluate the simulated scenario:

- satellite elevation and azimuth graphs.
- aircraft elevation and azimuth graphs.

#### Protocol

1. Run the ASE application.
2. Generate a ground station with a flight route.
3. Calculate the azimuth angle.
4. Display elevation and azimuth graphs.
5. Evaluate graphs.

#### Results

This experiment was repeated a few times with different input parameters. The graphs produced were then compared to graphs with similar parameters generated by a Matlab script. which was used in the system design phase of the project. The results obtained by comparing the graphs showed that the graphs matched, proving that the graphs were correctly generated by the ASE application.

## 6.4 SAA Control Module

The task of the following test is to verify the working of the SAA control module software, in particular the calculation of the steering angles.

### 6.4.1 Calculate Steering Angles

#### Objective

The objective of this test is to verify that the steering angles are correctly calculated. An arbitrary target ground station is chosen. A data file is then set up to provide the required input values instead of the ASE. The input parameters include the aircraft coordinates and attitude. The steering angles at the locations with the specified attitude provided by the data file are calculated beforehand. The steering angles calculated by the software must then be compared to those calculated beforehand.

#### Test Setup

- Load a data file with airborne object coordinates and attitude.
- Change the source of the input of the coordinates and attitude of the airborne object in the SAA control software from the ASE to the data file, which has been loaded.
- Run the SAA control software application.

#### Protocol

1. Perform the test setup.
2. Verify the calculated steering angles by reviewing the logged steering angles.

#### Results

The results concurred with the precalculated values.

#### Discussion

The results indicate that the steering angles are correctly calculated. Note, however, that the tests started with the placement of the airborne object, without changing the attitude, at various convenient locations relative to the ground station. For instance, at the same longitude as the ground station but at a different latitude. The tests then proceeded to more challenging locations. After these tests the influence of the attitude was taken into consideration. This was done with the same placement configuration as previously mentioned. The only difference is that the attitude of the airborne object was changed. The attitude was changed by changing the three attitude parameters, roll, pitch and yaw one by one and then finally changing all three at the same time.

## 6.5 ASE-Payload Interface Tests

The objectives of the following tests are to verify the communication interface between the ASE and the payload.

### 6.5.1 SAA Control Module Receive Telecommand Test

#### Objective

The objective of this experiment is to verify that the SAA control module is able to receive telecommands.

#### Test Setup

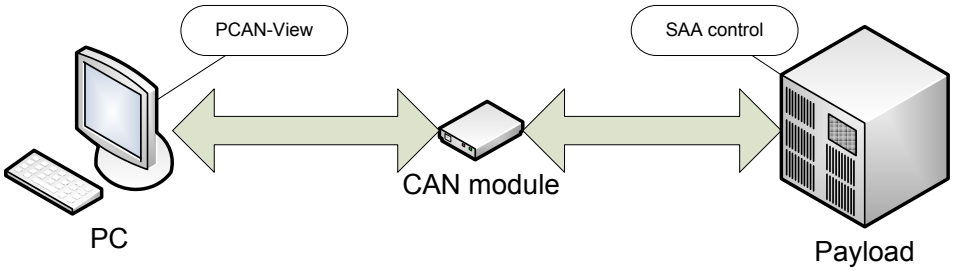


Figure 6.3: Setup for test 6.5.1

- Connect the PC with PCAN-View software to CAN module.
- Connect CAN module to CAN bus of payload.
- Run the PCAN-View application of the PC.
- Run the SAA control software application on payload.

#### Protocol

1. Perform the test setup.
2. Use PCAN-View to send a telecommand to the SAA Control module via the CAN bus.
3. Verify that the SAA control module has received a telecommand. A print statement in the code of the SAA control software is used to verify this. The PCAN-View application on the PC should also receive an acknowledge telecommand message from the SAA control software if the telecommand was received.

#### Results

The print statement printed the telecommand message when a telecommand was sent by the PCAN-View applicaton to the SAA control module. An *acknowledge* message was also received by the PCAN-View software after a telecommand was sent to the SAA control software.

#### Discussion

The *print* statements and the *acknowledge* messages verify that the SAA control module is able to receive telecommands successfully.

### 6.5.2 SAA Control Module Telemetry Request Test

#### Objective

This experiment's objective is to verify that the SAA control software can send a telemetry request.

#### Test Setup

- Repeat previous test setup (Setup of subsection 6.5.1)

#### Protocol

1. Perform the test setup.
2. Use the SAA control software to send a telemetry request.
3. Verify with PCAN-View that the SAA control software has sent a telemetry message.

#### Results

PCAN-View displays the telemetry message sent by the SAA control message.

#### Discussion

The message sent by the SAA control software is received by the PCAN-View software in the correct data format. This indicates that the SAA control software is able to send telemetry messages.

### 6.5.3 SAA Control Module Telemetry Reply Test

#### Objective

The objective of this experiment is to verify that the SAA control software can receive a telemetry reply that contains the requested data.

#### Test Setup

- Repeat previous test setup (Setup of subsection 6.5.1)

#### Protocol

1. Perform the test setup.
2. Use the SAA control software to send a telemetry request.
3. Use the PCAN-View software to reply to the telemetry request. The reply must contain some data.

4. Verify that the SAA control module has received telemetry data. A print statement in the code of the SAA control software is used to verify this.

#### Results

The telemetry data that was sent as a telemetry reply by the PCAN-View application was printed by the SAA control module.

#### Discussion

This test verified that the SAA control module is able to send a telemetry request and read the received data from the telemetry reply.

### 6.5.4 ASE Send Telecommand Test

#### Objective

The objective of this test is to verify that the ASE application on the PC is able to send telecommands.

#### Test Setup

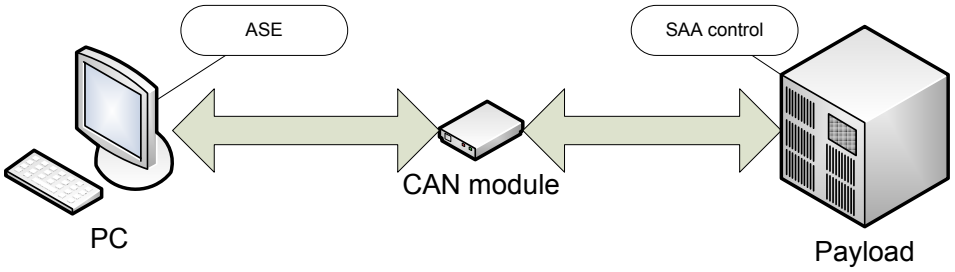


Figure 6.4: Setup for test 6.5.4

- Connect PC with ASE software to CAN module.
- Connect CAN module to CAN bus of payload.
- Run the ASE application of the PC.
- Run the SAA control software application on payload.

#### Protocol

1. Perform the test setup.
2. Use ASE software to send a telecommand to the SAA Control module via the CAN bus.
3. Verify that the SAA control module has received a telecommand. A print statement in the code of the SAA control software is used to verify this. The ASE software on the PC should also receive an *acknowledge telecommand* message from the SAA control software if the telecommand was received.

#### Results

The telecommand was printed when the SAA control software received a telecommand proving that the ASE software is able to send telecommands.

### 6.5.5 ASE Receive Telemetry Request Test

#### Objective

This experiment's objective is to verify that the ASE software is able to receive telemetry requests from the payload software.

#### Test Setup

- Repeat previous test setup (Setup of subsection 6.5.4)

#### Protocol

1. Perform the test setup.
2. Use SAA control software on the payload to send telemetry requests to the ASE software via the CAN bus.
3. Verify that the ASE software has successfully received the telemetry requests. This is achieved by the ASE software printing the received telemetry messages.

#### Results

The following telemetry messages were sent by the SAA control software and printed by the ASE software when received:

- 0x0A000A05
- 0x0A010A05
- 0x0A020A05

#### Discussion

The results indicate that the ASE is able to receive telemetry messages from the SAA control software.

### 6.5.6 ASE Telemetry Reply Test

#### Objective

The aim of this test is to verify that the ASE software is able to reply with the correct data to telemetry requests from the SAA software. The SAA software must also be able to read the data correctly from the telemetry replies received.

#### Test Setup

- Repeat previous test setup (the setup of subsection 6.5.4)

#### Protocol

1. Perform the test setup.
2. Use SAA control software on the payload to send telemetry requests to the ASE software via the CAN bus.
3. Verify that the ASE software has successfully replied to the telemetry requests. This is achieved by the SAA control software printing the telemetry reply messages received and the data that these messages contain.

#### Results

Various telemetry messages that requested data were sent to the ASE. The data that the ASE replied with was then displayed. The ASE replied with the data and the data was correctly read and displayed from these replies.

#### Discussion

The results showed that communication between the SAA control software and the ASE is successful. The SAA control software also retrieves this data correctly from the reply message.

## 6.6 ASE-Aircraft Avionics Equipment Interface Tests

### 6.6.1 Aircraft Avionics Equipment Emulator Test

#### Objective

The objective of this experiment is to verify that the working of the interface to the aircraft avionics equipment by testing the interface with the AAEE software that was developed for just such a purpose. The ASE is not connected to the actual avionics equipment, but to the AAEE, and therefore this experiment can be seen as the preliminary test in the quest to verify the working of this interface.

#### Test Setup

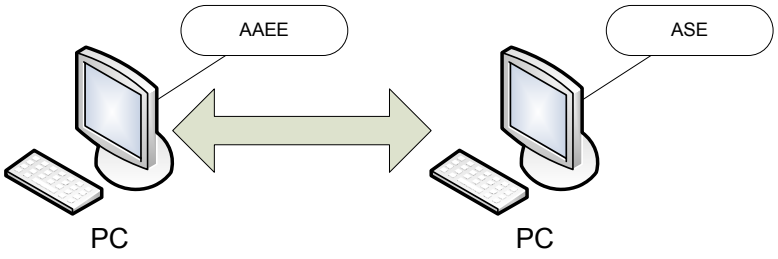


Figure 6.5: Setup for test 6.6.1

- Connect the serial ports of two PCs.
- On one PC run the ASE software.
- On the other PC run the aircraft avionics equipment emulator (AAEE) software.

**Protocol**

1. Perform the test setup.
2. Run the ASE application.
3. Initialise the serial port connected to the PC, using the AAEE software, by clicking on the corresponding com port button.
4. Run the AAEE application on the other PC.
5. Set up AAEE to transmit data.
6. Verify that the ASE software receives the aircraft avionics equipment data correctly from the AAEE. This can be done either by visually inspecting the avionics data displayed by the ASE as the AAEE transmits the data or by comparing the received avionics data logged by the ASE to the sent avionics data, also logged by the AAEE.

**Results**

By visual inspection and by comparing the logged files it was found that the data concurred.

**Discussion**

The results above indicate that this interface is working. Note, however, that this is only the preliminary test.

### 6.6.2 Aircraft Simulator Test

#### Objective

The objective of this test is to further verify that the serial interface to the aircraft avionics equipment is working before an actual flight test by connecting the ASE to the aircraft simulator.

#### Test Setup

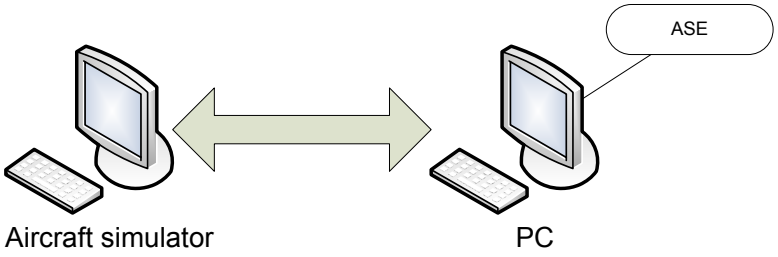


Figure 6.6: Setup for test 6.6.2

- Connect the serial port of the PC to the aircraft simulator.
- Run the ASE software on the PC.
- Set up the aircraft simulator to transmit data.

#### Protocol

1. Perform the test setup.
2. Run the ASE application.
3. Initialise the serial port connected to the AE by clicking on the corresponding com port button.
4. Run the AE software.
5. Set up AE to transmit data.

6. Verify that the ASE software receives the aircraft avionics equipment data correctly from the AE. This can be done either by visually inspecting the avionics data displayed by the ASE as the AE transmits the data or by reviewing the avionics data logged by the ASE.

#### Results

By visual inspection and by reviewing the logged files it was found that the data concurred.

#### Discussion

As with the previous test, the objective of this test is to verify that the interface to the avionics equipment works before an actual flight test. The result obtained indicates that this interface works and that the testing can proceed to flight tests.

## 6.7 SAA Control Module-Scheduling Software Tests

### 6.7.1 Scheduling Interface Test

#### Objective

The objective of this test is to see if the SAA control module is able to communicate with the scheduling software. This communication entails the request of the target ground station coordinates from the scheduling software and the reception of these coordinates.

#### Test Setup

- Set up a ground station schedule for the scheduling software.
- Run SAA control application on payload.
- Run scheduling software application on payload.

#### Protocol

1. Perform the test setup.

2. Confirm the reception of coordinates of target ground station.

#### Results

SAA control software confirms the reception of the target ground station coordinates by printing the ground station coordinates, once received, on the command line of the SH4. The coordinates received, that were printed, and the time interval between the target ground stations, concurred with the schedule that had been set up. This test was repeated a few time with different target ground station schedules. However, the results remained the same.

#### Discussion

The results of the tests indicate that the SAA control module correctly communicates with the scheduling software and receives the target ground station coordinates at the correct time.

## 6.8 Preliminary System Test

In order to verify the design and subsequent usability of the system, some preliminary tests were performed to evaluate the emulator system performance before an actual flight test. Two series of tests were performed, without the SAA. The tests used the AAEE and the aircraft simulator respectively, to provide aircraft avionics data to the system. The commands intended for the SAA were then captured from the payload and evaluated. The tests were performed as follows:

### 6.8.1 AAEE as Input

#### Objective

As a first attempt, the AAEE was used to produce realistic avionics flight data, as for an envisaged test flight. This was run on a separate CPU and the data was serially fed into the ASE in real time. The ASE output, in reaction

to the flight path as configured and simulated data as received, was fed onto the payload CAN bus for processing by the OBC. The latter was to produce the required steering commands for the SAA. The required steering angles, and commensurate commands, were calculated in advance using the orbital calculations as discussed earlier. The entire process was logged by means of suitable scripts and the output of the payload to the SAA was captured on a separate PC.

**Test Setup**

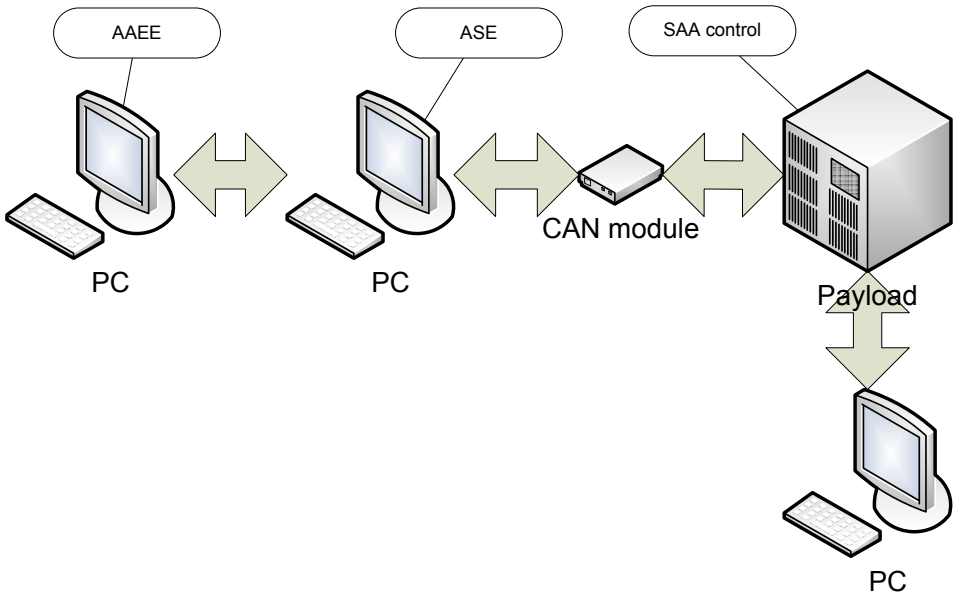


Figure 6.7: Preliminary system test set up with the AAEE

- Connect the PC with the AAEE software to the ASE PC.
- Connect the CAN module to the ASE PC and the CAN bus of the payload.
- Connect a PC, with the appropriate software, to the payload to capture the commands sent by the payload to the SAA.
- Run the SAA control software.
- Run the ASE software.

- Run the AAEE software.
- Run the software application that captures the commands intended for the SAA.

#### Protocol

1. Perform the test setup.
2. Set up the flight route on the ASE.
3. Set up the AAEE to fly on the specified flight route.
4. Evaluate the logged and captured data.

#### Results

Subsequent evaluation of the scripts and comparison of the actual steering commands with the previously calculated values, indicated no deviation of note and created confidence in an eventual closed loop performance.

#### Discussion

The results were considered positive, as no deviation from the previously calculated values occurred, which indicates that the second preliminary tests can proceed.

### 6.8.2 Aircraft Simulator as Input

#### Objective

As a second round of more stringent tests, approaching practical flight, a hardware based, standalone aircraft flight simulator was coupled to the ASE. The aircraft simulator closely models the light aircraft to be used in flight tests. The simulator is controlled by a joystick, pedals and other realistic hard interfaces, thus contributing the unknown quantity of pilot skill.

#### Test Setup

- Connect the aircraft simulator to the ASE PC.

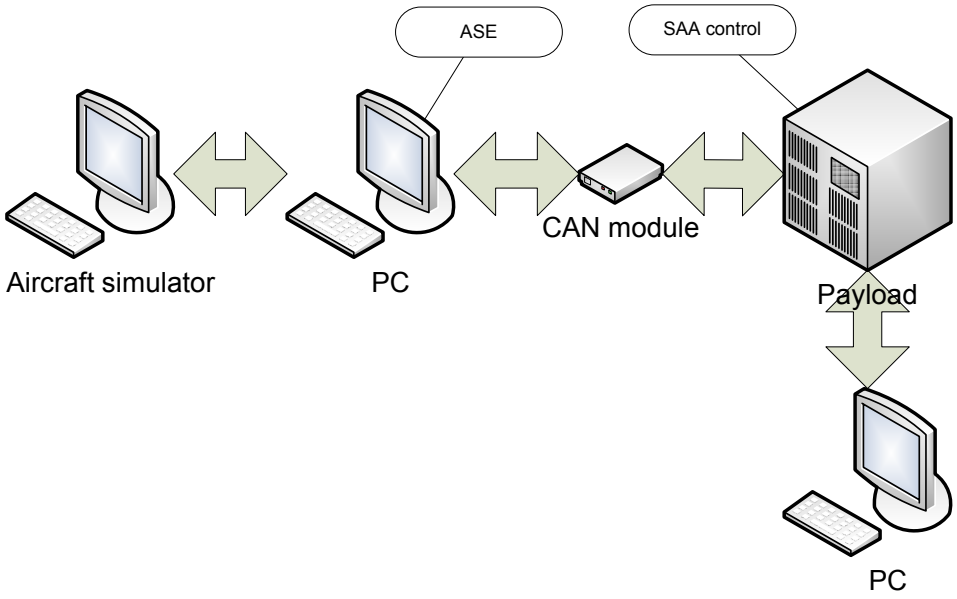


Figure 6.8: Preliminary system test set up with the aircraft simulator

- Connect the CAN module to the ASE PC and the CAN bus of the payload.
- Connect a PC with the appropriate software to the payload, to capture the commands sent by the payload to the SAA.
- Run the SAA control software.
- Run the ASE software.
- Run the aircraft simulator software.
- Run the software application that captures the commands intended for the SAA.

**Protocol**

1. Perform the test setup.
2. Set up the flight route on the ASE.
3. Fly the specified flight route with the aircraft simulator. See Figure 6.2 for a screenshot.
4. Evaluate the logged and captured data.

### Results

The results of this test are shown in Figures 6.9 to 6.14. These figures present three sets of data. (For the sake of clarity, some smaller portions of the graphs are shown magnified) The first two sets contain the predicted satellite and aircraft data. These were calculated with the help of calculations derived in Chapter 2. The third set shows the flight aircraft data as generated by a simulated aircraft flight.

The attitude of the aircraft is described by Figure 6.9. The figure indicates the difficulty experienced, in this case, in maintaining a constant attitude. It should be mentioned that the simulator was not flown on autopilot.

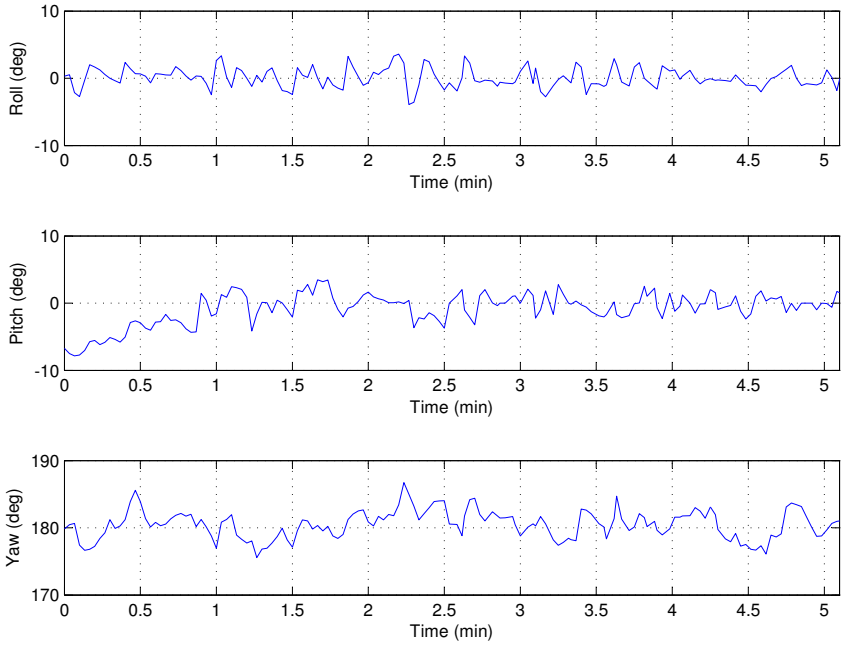


Figure 6.9: Measured aircraft flight test roll, pitch and yaw data

Figures 6.10 and 6.11 show the calculated elevation and azimuth angles. As can be seen from these two figures, the simulated flight data closely resembles that of the predicted aircraft flight. The elevation and azimuth angles are not much affected by the attitude, which is in line with the theoretical findings as presented earlier.

The calculated $\theta$ and $\phi$ angles are shown in Figures 6.12 and 6.13. A slight deviation is seen from the predicted aircraft data. This deviation is attributed

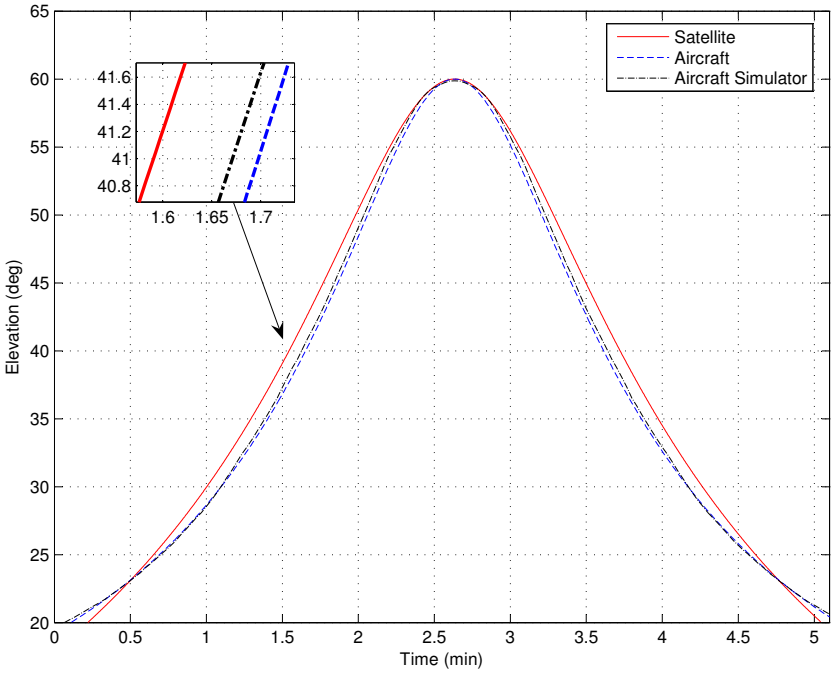


Figure 6.10: Calculated satellite data, predicted aircraft data and measured aircraft data for flight test elevation

to the varying attitude of the aircraft. It is because of this expected deviation that the first approach, mentioned in Section 3.1, was not chosen. A better approach is to emulate the elevation-azimuth angles more closely than the $\theta$-$\phi$ angles, that will vary in any case. The antenna control algorithm uses these $\theta$ and $\phi$ angles to beam steer when the antenna steering control algorithm compensates for this varying attitude, as it should. The varying attitude should not affect the link budget significantly.

### Discussion

The preliminary tests as mentioned above, have proved the functionality of the design and have not uncovered any reasons that the design should not perform as expected in the actual flight test.

## 6.9 Preliminary System Test with SAA

The following test was performed to evaluate the performance of the developed system with a prototype SAA, where the main focus was to see whether the two

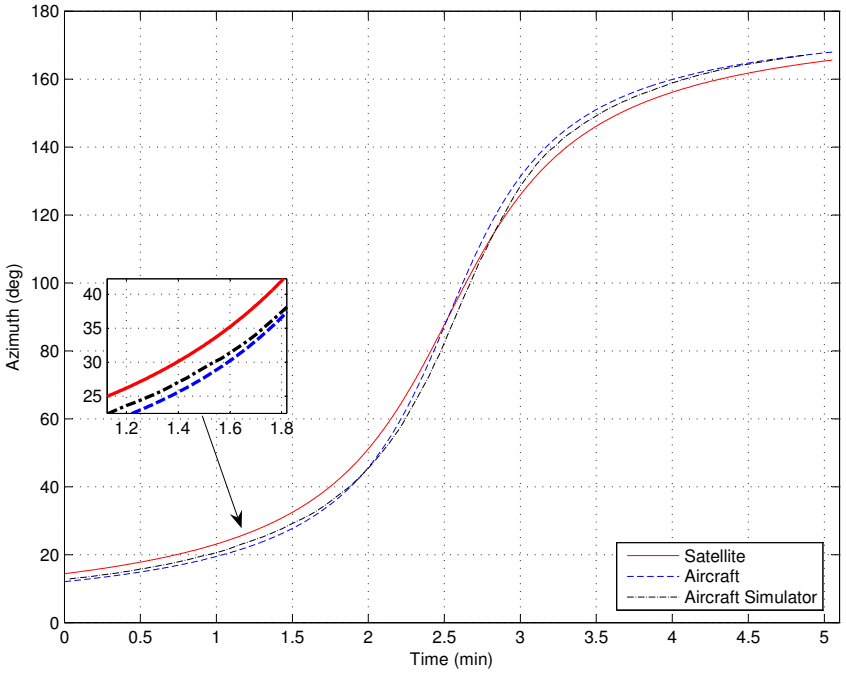


Figure 6.11: Calculated satellite data, predicted aircraft data and measured aircraft data for flight test azimuth

systems communicated correctly with each other. The tests were performed in conjunction with KU Leuven. As mentioned earlier in this thesis, KU Leuven developed the SAA, and the SAA used in this test was a prototype of theirs.

### Objective

The objective of the following test is first to confirm that the SAA control software on the payload can communicate with the SAA, then to verify that the SAA control software steers the SAA in the correct direction. To confirm the latter, the aircraft simulator was coupled to the system, a ground station was selected and the specified flight route flown.

A signal generator placed directly in front of the SAA provided a constant signal source to the SAA. A reduction of the received signal occurs when the SAA is steered. The SAA is steered according to where the ASE indicates the aircraft is on the flight route. Therefore, if the aircraft is directly above the ground station, no phase shift will be applied and the received signal will be at its strongest.

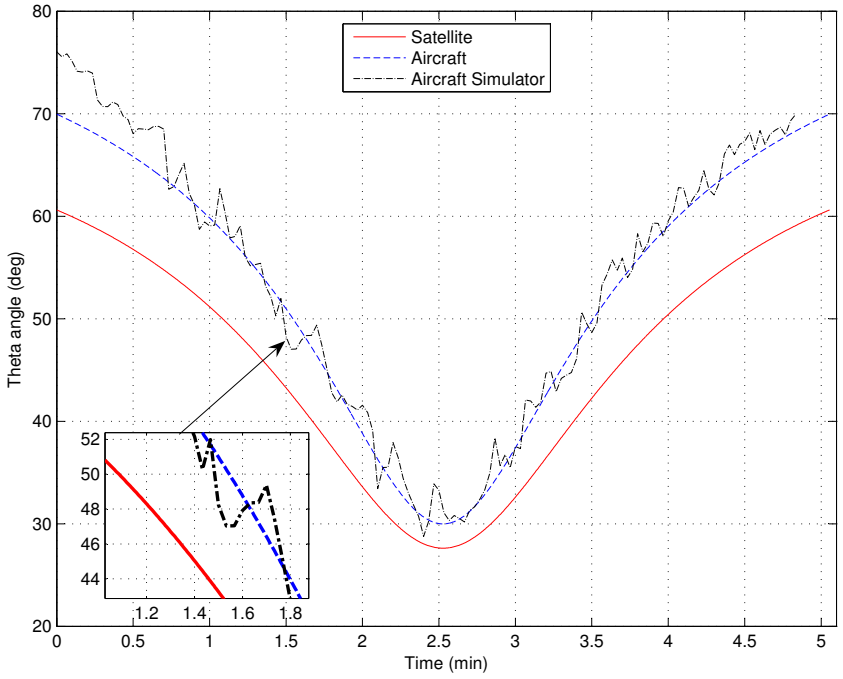


Figure 6.12: Calculated satellite data, predicted aircraft data and measured aircraft data for flight test $\theta$ angle

### Test Setup

- Connect the aircraft simulator to the ASE PC.
- Connect the CAN module to the ASE PC and the CAN bus of the payload.
- Connect the prototype SAA to the payload.
- Set up a signal generator directly in front of the SAA to provide a constant source to the SAA.
- Run the SAA control software.
- Run the ASE software.
- Run the aircraft simulator software.

### Protocol

1. Perform the test setup.

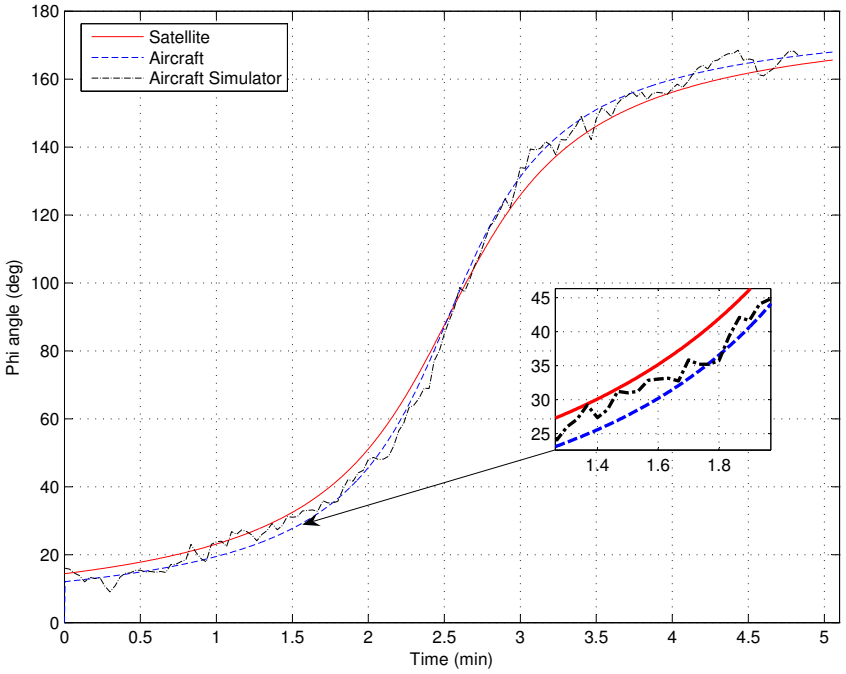


Figure 6.13: Calculated satellite data, predicted aircraft data and measured aircraft data for flight test $\phi$ angle

2. Setup the flight route on the ASE.
3. Fly the specified flight route with the aircraft simulator.
4. Confirm synchronisation.
5. Evaluate the logged and captured data.
6. Confirm steering angles.

### Results

As the objectives specify, the task is to verify that the payload software and the SAA can communicate with each other. The first step in this process is the synchronisation of the two devices. The SAA will send a specific byte back to the payload to indicate that the payload was able to successfully synchronise with the SAA. On the reception of the SAA control software on the payload will display a success message on the command line of the SH4. The synchronisation of the two devices was confirmed by the means mentioned above. The next step was to verify that the SAA control software could steer

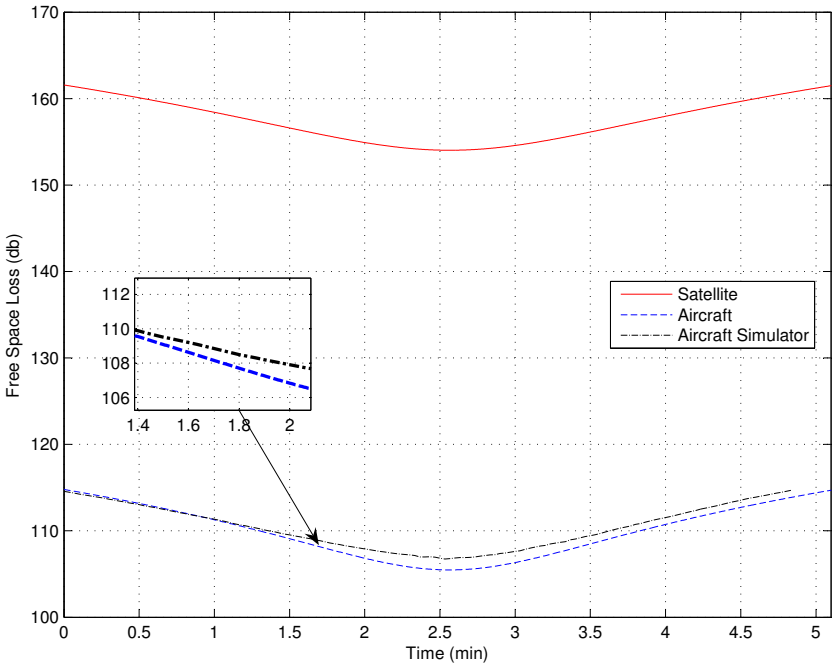


Figure 6.14: Calculated satellite data, predicted aircraft data and measured aircraft data for flight test free space loss

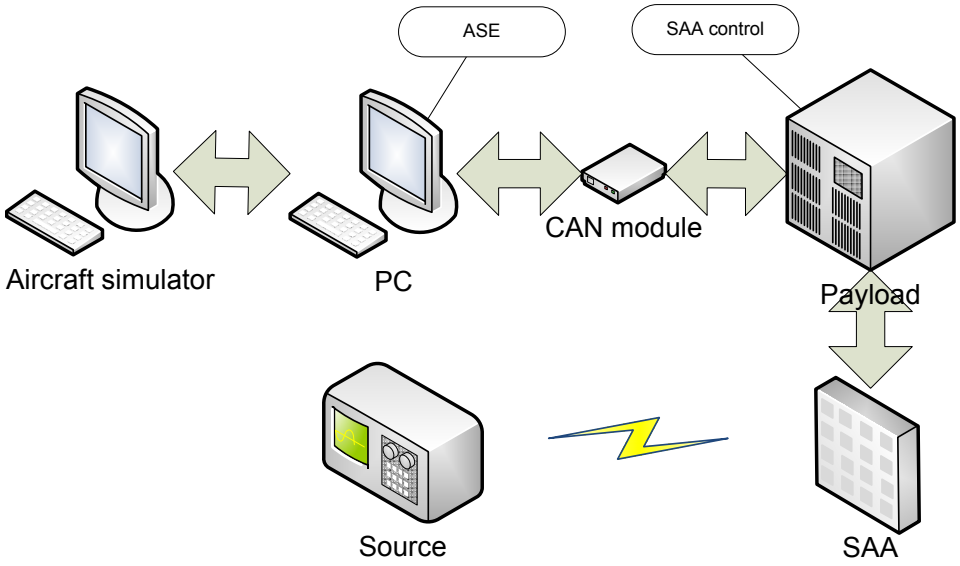


Figure 6.15: Test set up with SAA

the SAA correctly, which should further confirm that the two devices can communicate with each other.

Data captured from the SAA was used by KU Leuven to confirm that the

SAA was indeed steered correctly. The I and Q signals received from the individual antenna array elements, as well as the summation of all the Is and Qs after the phase shift were captured. KU Leuven used this data to confirm that the SAA control module on the payload controlled the SAA as expected to enable the SAA to phase shift the signal correctly in order to direct the antenna beam to a specific ground station. KU Leuven reviewed the data captured from the SAA and concluded that the SAA was steered according to the specified flight path. The communication between the payload and SAA was, therefore considered a success. Figure 6.16 shows a photo of the test setup.

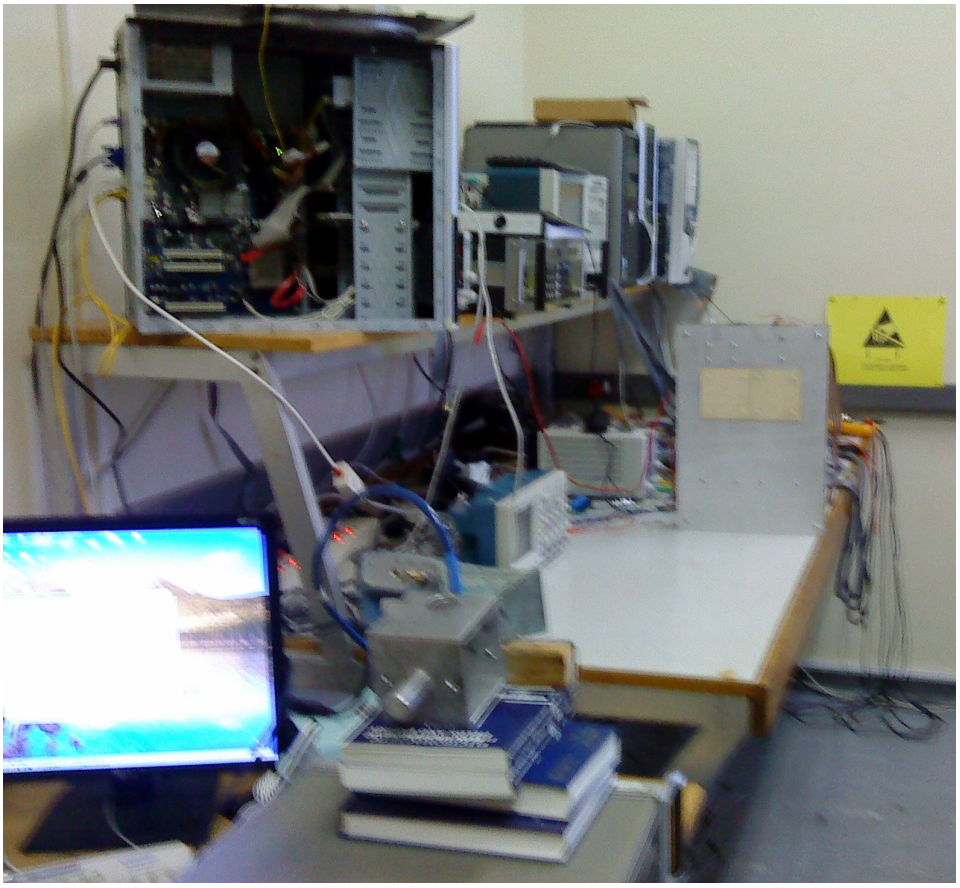

Figure 6.16: Photo of test set up with SAA

## 6.10 Conclusion

The initial results obtained by means of an aircraft simulator confirm the requirement to follow an exact flight path while maintaining a constant attitude. Any deviation will affect the performance of the emulation. However, the ground station elevation and azimuth angles are less affected than the satellite antenna $\theta$ and $\phi$ angles by small aircraft attitude fluctuations, for reason that the elevation and azimuth angles are measured from the perspective of the ground station and $\theta$ and $\phi$ from the aircraft.

Attitude changes obviously also change the orientation of the antenna on the aircraft. This, however, will be compensated for by the antenna control algorithm in correcting the $\theta$ and $\phi$ steering angles (Figures 6.12 and 6.13). The effect on the transmission link is, therefore, relatively small, apart from some required change in the induced free space loss (Figure 6.14).

During the simulator test, an exact flight path presented some difficulty to the particular simulator pilot, but one would expect an experienced pilot, assisted by autopilot, to fare better in actual flight. This would be essential.

Furthermore, results from the previous section indicate that deviations occur at low elevation angles. A flight path with a higher maximum elevation angle will, therefore, emulate the satellite more accurately. This is not seen as a major drawback, as the emulation time window is still adequate.

The logs from the tests as performed also indicate that the beam steering commands were correctly generated by the OBC in response to the ASE inputs, which were processed from aircraft avionics inputs, flight path data as defined and ground station coordinates.

The initial results clearly indicate that it is quite feasible to use a light aircraft equipped with a suitable emulator, to act as an initial flight test platform, in order to evaluate the performance of the beam steerable satellite antenna and peripheral associated payload components. Although many other factors, such as overall hardware space endurance etc. enter into the design of actual space flight hardware, the economics and convenience of the approach, as set out, are beyond question. It was also proved that an actual LEO satellite flight path could be emulated to an acceptable degree. The ASE with incumbent software as developed, will furthermore act as a realistic and flexible interface between the aircraft and the satellite payload under development.

Results from tests conducted with an actual hardware based, standalone flight simulator proved the accuracy and functionality of the ASE.

The tests performed with the prototype SAA further proved the functionality of the interface between the two systems works and that the software on the payload is able to control the SAA as required. In summary, no indication was found that the emulator could not be practically be employed as intended.

# Chapter 7

# Conclusion

## 7.1 Summary of the Work Conducted

This thesis has described the development of a simple LEO satellite emulation system for the purpose of conveniently testing a satellite payload with SAA. Five main objectives were set at the start of the project. These can be restated as part of the following summary:

Chapter 2 defined the concepts required for system development. These included the development of a LEO satellite model as used throughout this project to investigate the orbital characteristics of a LEO satellite. This background was essential to achieve the first objective of a suitable emulation strategy, as detailed in Chapter 3.

This emulation strategy comprised a flight and transmission link strategy respectively. The chosen flight strategy is for an aircraft to fly past a ground station in a straight path parallel to the surface of the earth at a specific constant speed and altitude, thereby emulating the time variant elevation-azimuth angles of a LEO satellite. The transmission link strategy requires attenuation of the received or transmitted signal at the ground station to compensate for the free space losses.

The next three objectives achieved were all described in Chapters 4 and 5. These goals constitute the design and implementation of an emulation system, a control module to control the SAA and the successful integration of all the interfaces. In Chapter 4 the two main components of the system design, ie. the ASE and the SAA control modules with their functional sub-units, were presented and discussed. The ASE calculates the required aircraft

parameters emulating a LEO satellite and generates the necessary data for the flight payload. The SAA control module is housed on the payload OBC and controls the SAA. The system design also defines the component interface requirements. The implementation of the system was covered in Chapter 5.

Chapter 6 documented the successful evaluation of the system. The evaluation strategy started by testing and debugging all the smaller functions before proceeding to the component interface tests. Once all these tests had been successfully completed, the focus shifted to the whole system.

## 7.2 Contributions

Apart from meeting the objectives as originally set out, the following contributions were also made during that process:

- A required flight profile for a relatively slow flying aircraft, to emulate a LEO circular orbit trajectory, were investigated in detail.
- An aircraft based emulation strategy to emulate LEO satellite orbits, was defined and developed.
- A practical, general LEO emulation platform (ASE) that could be used for aircraft borne LEO payload testing, was developed. This development is not only suitable for the SAA payload in question, but can be adapted for testing various satellite payloads. Such an approach is a flexible and clearly cost effective means of actual preflight system testing.
- A beam steering strategy was developed on the payload to direct the SAA towards a specific target.
- In order to acieve this, the following interfaces were created:
  - Aircraft avionics equipment and emulation platform (ASE) interface.
  - Interface between the emulation platform (ASE) and satellite bus.
  - Satellite payload and SAA control interfaces.
- The above were satisfactorily proved by thorough ground based testing and evaluation of the system.

## 7.3 Recommendations

Recommendations to complete the project and possibly improve system performance, can be made as follows:

- The recommendation to proceed to an actual flight test with fully integrated system, is based on the successful results obtained from the ground based testing with the aircraft simulator. This particular test was at the time of writing not possible, due to uncompleted logistics between KU Leuven, developing the SAA, and Stellenbosch University developing the payload.
- The ASE software could possibly be modified and made still more user friendly, after some actual flight experience.
- The ASE software could be modified to be compatible with other operating systems.
- The terrain map could be improved. This could be achieved by importing an actual photo of the terrain, or by displaying the map in a 3-D view, providing the user with a better altitude perspective.
- Playback of a flight currently occurs in real time. A capability can be added to fast forward the playback, thereby reducing playback time. This could be a useful user feature.
- The position of the aircraft is currently displayed as a dot on the terrain map. An improvement would be to add a feature to display the aircraft attitude. This information would provide the user with a better situational awareness.
- The direction of the antenna beam could be displayed on the ASE terrain map. This could be achieved by forwarding the steering angles calculated by the payload OBC software, to the ASE through the CAN bus. Displaying the antenna beam will allow the user to immediately verify the correctness of the calculated steering angles.
- The overall memory usage of the software could be improved.

- The execution speed of the software could be improved, reducing the processing load on the OBC. Using sine and cosine look up tables and interpolating software, overhead could be reduced.

# Appendices

# Appendix A

# Transmission Link

Uplink Performance

| Uplink frequency (MHz) | 2400.0000 |
|---|---|
| GS TX power (W) | 10.0 |
| GS TX antenna gain (dBi) | 7.0 |
| Satellite RX antenna gain (dBi) | 8.0 |
| Receiver noise floor (dBm) | -112.0 |
| Minimum required RX SNR (dB) | 0.0 |
| Polarisation mismatch losses (dB) | 3.0 |

| Elevation (degrees) | 0.0 | 5.0 | 10.0 | 15.0 | 20.0 | 25.0 | 30.0 | 35.0 | 40.0 | 45.0 | 50.0 | 55.0 | 60.0 | 65.0 | 70.0 | 75.0 | 80.0 | 85.0 | 90.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Slantrange (km) | 2573.1 | 2077.1 | 1694.6 | 1407.2 | 1192.8 | 1031.8 | 909.4 | 815.0 | 741.3 | 683.1 | 636.8 | 599.9 | 570.5 | 547.4 | 529.5 | 516.3 | 507.1 | 501.8 | 500.0 |
| Satellite angle | 68.0 | 67.5 | 65.9 | 63.6 | 60.6 | 57.2 | 53.4 | 49.4 | 45.3 | 41.0 | 36.6 | 32.1 | 27.6 | 23.1 | 18.5 | 13.9 | 9.3 | 4.6 | 0.0 |
| Pathloss (dB) | 168.3 | 166.4 | 164.6 | 163.0 | 161.6 | 160.3 | 159.2 | 158.3 | 157.4 | 156.7 | 156.1 | 155.6 | 155.2 | 154.8 | 154.5 | 154.3 | 154.1 | 154.1 | 154.0 |
| Received signal power (dBm) | -116.3 | -114.4 | -112.6 | -111.0 | -109.6 | -108.3 | -107.2 | -106.3 | -105.4 | -104.7 | -104.1 | -103.6 | -103.2 | -102.8 | -102.5 | -102.3 | -102.1 | -102.1 | -102.0 |
| Receiver signal to noise ratio (dB) | -4.3 | -2.4 | -0.6 | 1.0 | 2.4 | 3.7 | 4.8 | 5.7 | 6.6 | 7.3 | 7.9 | 8.4 | 8.8 | 9.2 | 9.5 | 9.7 | 9.9 | 9.9 | 10.0 |
| Link margin (dB) | -4.3 | -2.4 | -0.6 | 1.0 | 2.4 | 3.7 | 4.8 | 5.7 | 6.6 | 7.3 | 7.9 | 8.4 | 8.8 | 9.2 | 9.5 | 9.7 | 9.9 | 9.9 | 10.0 |

| Orbit (km) | 500.0 |
|---|---|
| Boltzman's constant k | 0 |

Figure A.1: Satellite up-link link budget

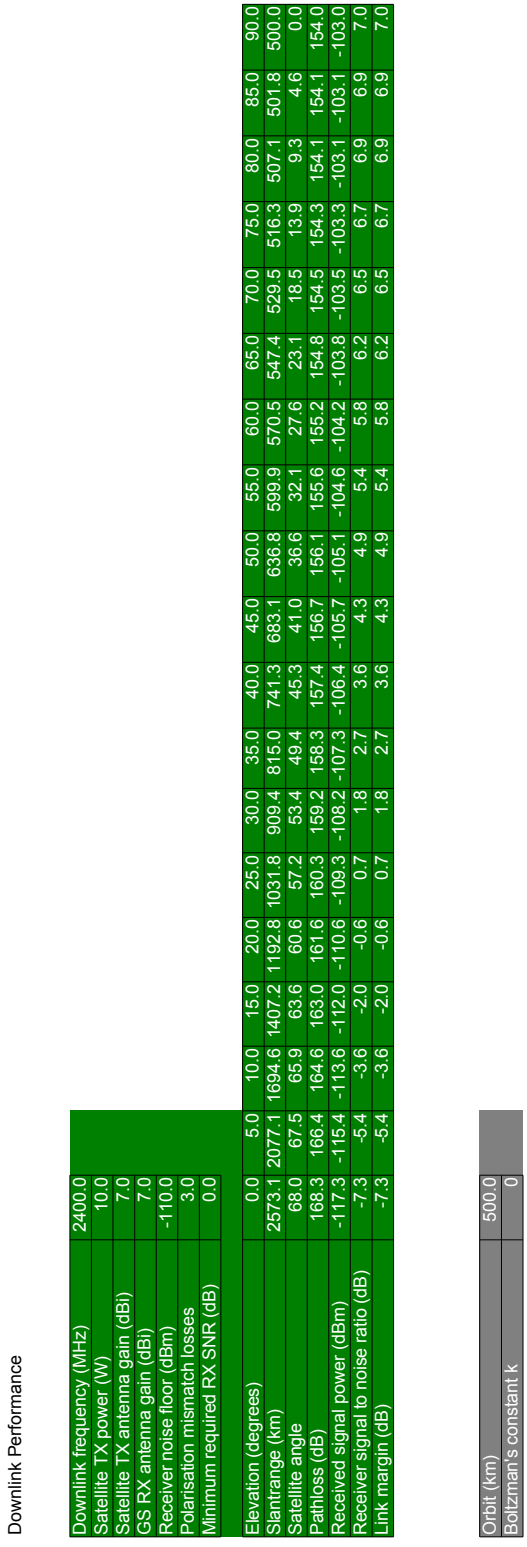


Figure A.2: Satellite down-link link budget

# List of References


[1] Aerts, W., Delmotte, P. and VandenBosch, G.: Conceptual study of analog baseband beam forming: Design and measurement of an eight-by-eight phased array. *IEEE transactions on antennas and propagation*, vol. 57, pp. 1667–1672, 2009.

[2] Nasa history office web page. 2010.
Available at: `http://history.nasa.gov/SP-367/appendc.htm`

[3] Koks, D.: Using rotations to build aerospace coordinate systems. Tech. Rep., Australian Department of Defence: Electronic Warface and Radar Division System Sciences Laboratory, 2006. Page 2.

[4] Koks, D.: Using rotations to build aerospace coordinate systems. Tech. Rep., Australian Department of Defence: Electronic Warface and Radar Division System Sciences Laboratory, 2006. Pages 2 - 3.

[5] Koks, D.: Using rotations to build aerospace coordinate systems. Tech. Rep., Australian Department of Defence: Electronic Warface and Radar Division System Sciences Laboratory, 2006. Page 4.

[6] Direction cosine matrix ecef to ned.
Available at: `http://www.mathworks.com/access/helpdesk/help/toolbox/aeroblks/directioncosinematrixeceftoned.html`

[7] Wertz, J. (ed.): *Spacecraft Attitude Determination and Control.* Kluwer Academic, 1978. Page 764.

[8] Larson, W. and Wertz, J. (eds.): *Space Mission Analysis and Design.* Microcosm, Inc. and Kluwer Academic Publishers, 1995. Pages 108 - 109.

[9] Charlambous, D.: Mathematical tools (physics studies). Tech. Rep., Department of physics Lancaster University. Page 4.

[10] The basics of satellite orbits.
Available at: `http://www.amacad.org/Publications/Section\_4.pdf`

[11] Campbell, B.A. and McCandlesss, S.W.: *Introduction to Space Sciences and Spacecraft Applications.* Gulf Publishing Company, 1996.

[12] Larson, W. and Wertz, J. (eds.): *Space Mission Analysis and Design.* 2nd edn. Microcosm, Inc. and Kluwer Academic Publishers, 1995.

[13] Maral, G. and Bousquet, M.: *Satellite Communications Systems.* 2nd edn. John Wiley & Sons Ltd, 1993.

[14] International telecommunication union. February 2009.
Available at: `http://www.itu.int`

[15] Aliakbarian, H., Volski, V. and VandenBosch, G.A.E.: The maximum gain of the antanna.

[16] Wi fi view.
Available at: `http://www.wifiview.com`

[17] Relative velocity and the doppler effect.
Available at: `http://departments.colgate.edu/physics/research/PhysicsEd/Lab4doppler.doc`

[18] Cakaj, S., Keim, W. and Malaric, K.: Communications duration with low earth orbiting satellites. In: *IASTED International Conference.* Montreal, May 30 - June 1 2007.

[19] Borland software "c++ builder". May 2010.
Available at: `http://www.borland.com/cbuilder`

[20] Helderberg aviation. December 2009.
Available at: `http://www.helderaviation.co.za`

[21] Hough, W.J.: *Autonomous Aerobatic Flight of a Fixed Wing Unmanned Aerial Vehicle.* Master's thesis, Stellenbosch University, 2007. Pages 142 - 144.

[22] Peddle, I.K.: *Autonomous Flight of a Model Aircraft.* Master's thesis, University of Stellenbosch, 2005. Pages 159 - 162.

[23] Dreijer, G.: Generic specification: Can node. Tech. Rep. 112-0000100-02C, SunSpace, 2008.

[24] Dreijer, G.: Generic specification: Sunspace can protocol. Tech. Rep. 112-0000150-00F, SunSpace, 2008.

[25] Can specification version 2.0. January 2009.
Available at: `http://www.semiconductors.bosch.de/pdf/can2spec.pdf`

[26] Road vehicles - controller area network (can) - part 2: High-speed medium access unit. January 2009.
Available at: `http://www.investigacion.frc.utn.edu.ar/sensors/Aplications/Divulgucion/c\&s/files/ISOCD11898-1.pdf`

[27] Road vehicles - controller area network (can) - part 1: Data link layer and physical signalling. January 2009.
Available at: `http://www.investigacion.frc.utn.edu.ar/sensors/Aplications/Divulgucion/c\&s/files/ISOCD11898-2.pdf`

[28] cplusplus.com. November 2009.
Available at: `http://www.cplusplus.com/reference/stl/list/`

[29] Koks, D.: Using rotations to build aerospace coordinate systems. Tech. Rep., Australian Department of Defence: Electronic Warface and Radar Division System Sciences Laboratory, 2006.

[30] Thompson, A.: .
Available at: `http://atacolorado.com/eulersequences.doc`